

LABORATÓRIO DE MEIOS POROSOS E
PROPRIEDADES TERMOFÍSICAS
e
NÚCLEO DE PESQUISA EM CONSTRUÇÃO

*Apostila de Programação Orientada a Objeto em C++
Parte V- Programação para Linux*



André Duarte Bueno, UFSC-LMPT-NPC

<http://www.lmpt.ufsc.br/~andre>

email: andre@lmpt.ufsc.br

Versão 0.4.5

12 de dezembro de 2002



Apostila de CVS. Versão 0.4.5

Distribuída na forma GFDL (<http://www.gnu.org/licenses/licenses.html#TOCFDL>).

Copyright (C) 2002 - **André Duarte Bueno**.

Esta apostila é “textbook” livre; você pode redistribuí-la e/ou modificá-la sob os termos da Licença Pública de Documentos da GNU (GFDL), conforme publicada pela Free Software Foundation; versão 1.2 da Licença como (a seu critério) qualquer versão mais nova; preservando as seções “no Invariant Sections no Front-Cover Texts, and no Back-Cover Texts”. Uma cópia da licença é localizada no capítulo *GNU Free Documentation License*. Esta apostila é “software” livre; você pode redistribuí-la e/ou modificá-la sob os termos da Licença Pública Geral GNU, conforme publicada pela Free Software Foundation; tanto a versão 2 da Licença como (a seu critério) qualquer versão mais nova.

Desenvolvida no

Laboratório de Meios Porosos e Propriedades Termofísicas (<http://www.lmpt.ufsc.br>)

e no Núcleo de Pesquisa em Construção (<http://www.npc.ufsc.br>), com apoio do

Curso de Pós-Graduação em Engenharia Mecânica (<http://www.posmec.ufsc.br>)

e da Universidade Federal de Santa Catarina (<http://www.ufsc.br>).

Sumário

I	Programação Para GNU/Linux, Unix	9
1	Introdução a Programação GNU/Linux, Unix	10
1.1	Termos e conceitos utilizados	11
1.2	O básico do GNU/Linux, Unix	12
1.2.1	Comandos do shell úteis	12
1.2.2	Expressões regulares ³	19
1.2.3	Programas telnet e ftp	20
1.3	Diretórios úteis para programadores	22
1.4	Programas úteis para programadores	23
2	Edição de Texto Emacs e VI	25
2.1	Introdução ao editor emacs	25
2.2	Comandos do editor emacs	25
2.2.1	Help	25
2.2.2	Movimento do cursor (use as setas de direção)	26
2.2.3	Cut/Copy/Paste/Undo	26
2.2.4	Arquivos	27
2.2.5	Pesquisa e substituição	27
2.2.6	Múltiplas janelas	27
2.2.7	Encerrando seção do Emacs	27
2.3	Comandos do editor vi	28
3	Os programas diff, patch, indent	30
3.1	O programa diff	30
3.1.1	Sentenças para o diff	32
3.2	O programa patch	32
3.3	O programa indent	33
4	Compilando com gcc, g++	36
4.1	Protótipo e parâmetros do gcc/g++	36
4.2	Arquivos gerados pelo gcc/g++	37
4.3	Exemplo de uso do gcc/g++	37
4.4	Sentenças para o compilador da gnu	38

5	Make	39
5.1	Um arquivo de projeto	39
5.2	Protótipo e parâmetros do make	39
5.3	Formato de um arquivo Makefile	40
5.3.1	Criando variáveis em um arquivo Makefile	40
5.3.2	Criando alvos em um arquivo Makefile	41
5.4	Exemplo de um arquivo Makefile	41
5.5	Sentenças para o make	43
6	Bibliotecas	44
6.1	Introdução a montagem de bibliotecas	44
6.1.1	ar	44
6.1.2	ranlib	45
6.1.3	nm	46
6.1.4	objdump	47
6.1.5	ldd	47
6.1.6	ldconfig	48
6.2	Convenção de nomes para bibliotecas	48
6.3	Bibliotecas usuais	48
6.4	Montando uma biblioteca estática (libNome.a)	48
6.4.1	Usando uma biblioteca estática	49
6.5	Montando uma biblioteca dinâmica (libNome.so)	50
6.5.1	Usando uma biblioteca dinâmica	51
6.5.2	Vantagens/desvantagens da biblioteca dinâmica	51
6.6	Sentenças para bibliotecas	54
7	Libtool	55
7.1	Introdução ao libtool ²	55
7.2	Forma de uso do libtool	55
7.3	Criando uma biblioteca sem o libtool	56
7.4	Criando uma biblioteca estática com o libtool	56
7.5	Criando uma biblioteca dinâmica com o libtool	57
7.6	Linkando executáveis	57
7.7	Instalando a biblioteca	57
7.8	Modos do libtool	58
7.9	Sentenças para o libtool	58
8	Debug (Depuradores, Debuggers)	60
8.1	Introdução ao debugger	60
8.2	Comandos do gbd	60
8.3	Exemplo de uso do gdb	60
8.4	Sentenças para o gdb	61
9	Profiler (gprof)²	62
9.1	Introdução ao gprof	62
9.2	Sentenças para o profiler:	62

10	Versão de Depuração, Final e de Distribuição	64
10.1	Versão debug, release e de distribuição	64
10.1.1	Versão debug	64
10.1.2	Versão final (release)	64
10.1.3	Distribuição dos programas e bibliotecas	64
10.2	Sentenças para distribuição de código fonte	67
11	Documentação de Programas Usando Ferramentas GNU/Linux	68
11.1	Introdução a documentação de programas	68
11.2	Documentação embutida no código com JAVA_DOC	68
11.2.1	Exemplo de código documentado	68
11.2.2	Sentenças para documentação java_doc	69
11.3	Tutorial de configuração e uso do DOXYGEN	70
11.3.1	Exemplo de programa documentado	72
11.3.2	Exemplo de diagramas gerados pelo doxygen	74
11.4	Documentação profissional com sgml/xml (LyX)	75
12	Seqüência de Montagem de Um Programa GNU²	78
12.1	Introdução a programação multiplataforma com GNU	78
12.2	aclocal	80
12.3	ifnames	80
12.4	autoscan	80
12.4.1	Roteiro do autoscan	81
12.5	autoheader	81
12.5.1	Roteiro do autoheader	81
12.6	automake	82
12.6.1	Introdução ao automake	82
12.6.2	Sentenças para o automake	85
12.7	autoconf	85
12.7.1	Introdução ao autoconf	85
12.7.2	Protótipo do autoconf	86
12.7.3	Roteiro do autoconf	86
12.7.4	Estrutura de um arquivo configure.in	86
12.7.5	Exemplo de um arquivo configure.in	87
12.7.6	Macros do autoconf	87
12.7.7	Como aproveitar os resultados das pesquisas realizadas pelo autoconf	90
12.7.8	Variáveis definidas no arquivo configure.in e que serão substituídas no arquivo Makefile	92
12.8	autoreconf	93
12.9	./configure	94
12.10	Como incluir instruções do libtool em seu pacote gnu	94
12.10.1	Exemplo de arquivo makefile.am usando o libtool	95
12.10.2	Exemplo de arquivo configure.in usando o libtool	95
12.11	Exemplo Completo	95

13 Introdução ao Controle de Versões Com o CVS	100
13.1 O que é o CVS?	100
13.2 Comandos do cvs	101
13.3 Sequência de trabalho	103
13.3.1 Roteiro para criar um repositório	103
13.3.2 Para importar os arquivos de seu projeto antigo para dentro do repositório	104
13.3.3 Para baixar o projeto	105
13.3.4 Para criar módulos	106
13.3.5 Para adicionar/remover arquivos e diretórios	107
13.3.6 Para atualizar os arquivos locais	110
13.4 Versões, tag's e releases	111
13.4.1 Entendendo as versões	111
13.4.2 Para criar tag's	111
13.4.3 Para criar release's	113
13.4.4 Recuperando módulos e arquivos	114
13.5 Para verificar diferenças entre arquivos	115
13.6 Verificando o estado do repositório	115
13.6.1 Histórico das alterações	116
13.6.2 Mensagens de log	116
13.6.3 Anotações	117
13.6.4 Verificando o status dos arquivos	117
13.7 Ramos e Misturas (Branching and Merging)	118
13.7.1 Trabalhando com ramos	118
13.7.2 Mesclando 2 versões de um arquivo	119
13.7.3 Mesclando o ramo de trabalho com o ramo principal	119
13.8 Configuração do cvs no sistema cliente-servidor ³	120
13.8.1 Variáveis de ambiente	120
13.9 Como baixar programas de terceiros usando o cvs	121
13.10 Frontends	121
13.11 Sentenças para o cvs	121
13.12 Um diagrama com os comandos do cvs	124

Lista de Figuras

2.1	O editor de texto emacs (observe o ítem de menu C++).	29
11.1	Ilustração da hierarquia TMatriz da biblioteca LIB_LMPT.	74
11.2	Ilustração da hierarquia da classe TRotulador3D da biblioteca LIB_LMPT.	75
11.3	Ilustração das dependências do arquivo TRotulador3D.	76
11.4	O site do LyX.	77
13.1	Versões de um arquivo.	111
13.2	Criando um tag.	112
13.3	Criando um release.	113
13.4	Como ficam os ramos.	118
13.5	Um frontend para o cvs no GNU/Linux, Unix (o cervisia).	122
13.6	Um frontend para o cvs no Windows.	123
13.7	Diagrama com os comandos do cvs.	124

Lista de Tabelas

1.1	Diretórios importantes para o programador.	22
1.2	Programas úteis para desenvolvedores de software no ambiente GNU/Linux.	24
6.1	Bibliotecas usuais.	49
8.1	Comandos do gdb.	61
12.1	Sequência executada pelo programador para montagem de programas multi-plataforma usando o padrão GNU.	79
12.2	Sequência executada pelo usuário.	79

Listings

3.1	Arquivo e06a-hello.cpp.	30
3.2	Arquivo e06b-hello.cpp.	31
3.3	Arquivo diff.	31
3.4	Arquivo diff -c.	31
3.5	Arquivo diff -u.	32
3.6	Arquivo ex-vector-1-indent.cpp.	34
5.1	Arquivo makefile.	41
5.2	Exemplo de uso do programa make.	42
6.1	Saída do comando ar - -help.	45
6.2	Saída do comando nm -help	46
6.3	Saída do comando ldd /usr/bin/lyx.	47
6.4	Arquivo makefile com bibliotecas estáticas e dinâmicas.	51
6.5	Arquivo mostrando o uso do makefile.	53
7.1	Arquivo libtool - -help.	55
11.1	Exemplo de arquivo de configuração do doxygen com comentários.	71
11.2	Saída do comando doxygen - -help.	72
11.3	Exemplo de código documentado no formato JAVA_DOC para uso com o programa doxygen.	72
13.1	Saída do comando: cvs - -help-options	101
13.2	Saída do comando: cvs - -help-commands	101
13.3	Saída do comando: cvs- -help-synonyms	102
13.4	Saída do comando: cvs import	104
13.5	Como fica o repositório após a importação	105
13.6	Saída do comando: cvs -H checkout	105
13.7	Saída do comando: cvs -H commit	107
13.8	Saída do comando cvs commit após adição de um módulo	107
13.9	Saída do comando: cvs -H update	110
13.10	Saída do comando: cvs -tag nome	112
13.11	Saída do comando: cvs commit -r 2	114
13.12	Saída do comando: cvs-diff	115
13.13	Saída do comando: cvs -log leia.me.txt	116
13.14	Saída do comando: cvs -status leia.me.txt	117
	Lista de programas	

Parte I

Programação Para GNU/Linux, Unix

Capítulo 1

Introdução a Programação GNU/Linux, Unix

Esta parte da apostila contém dicas e instruções para montagem de programas usando o formato multiplataforma da GNU. Inicialmente apresenta-se uma lista de comandos do shell e de programas úteis no ambiente GNU/Linux. Descreve-se o programa, sua utilidade, os arquivos de configuração e os parâmetros opcionais. Quando conveniente apresenta-se um exemplo.

A seguir descreve-se o uso dos editores emacs e vi, e os programas diff, patch e indent. Descreve-se o compilador gcc/g++ e o make para montagem de programas pequenos. Como montar bibliotecas, o uso do debugger e do profiler, como distribuir seu código, como documentar seus programas.

Depois, apresenta-se um roteiro para montar programas completos usando o padrão GNU.

- O texto aqui apresentado é um texto introdutório.
- Um texto intermediário, que abrange diversos aspectos da programação para GNU/Linux (ferramentas da gnu, processos, sinais, device drives, programação em ambientes gráficos) é encontrado no livro “*Linux Programming Unleashed*” de Kurt Wall *et al.* Em português existe o “*Programando com Ferramentas GNU*” (editado pela conectiva).
- Textos avançados são os manuais disponibilizados na internet, isto é, para aprender em detalhes o autoconf, baixe o manual do autoconf, o mesmo é válido para os demais tópicos apresentados nesta apostila.

Este resumo tem como base as páginas man e os manuais públicos do make (3.78), egcs (1.1.2), egcs++ (1.1.2), autoconf (2.13), automake (1.4), libtool¹, o manual do doxygen e artigos da revista do GNU/Linux (<http://www.revistadolinux.com.br>). Também foram consultadas as referências [13, 9, 12, 11, 3] [2, 10, 1, 4] [8] [7, 5].

¹Estes manuais são encontrados, em inglês, no site da gnu (<http://www.gnu.org>).

1.1 Termos e conceitos utilizados²

Apresenta-se aqui, brevemente, uma explicação dos termos utilizados no mundo do software livre.

Linux: clone livre do kernel do UNIX, escrito a partir do zero por Linus Torvalds, com a ajuda de um grupo de programadores espalhados pela internet [isto é, o linux é somente um KERNEL]. Foi projetado para estar em conformidade com o POSIX e com a Single Unix Specification.

GNU: acrônimo para “GNU não é UNIX”. GNU é o nome de um sistema operacional completo e compatível com UNIX escrito a partir de 1983 por Richard Stallman e inúmeros hackers da comunidade de software livre espalhados pela internet. O GNU é um sistema 100% livre, ou seja, ele fornece as 4 liberdades básicas do software livre: A liberdade de uso, modificação, cópia e publicação de versões modificadas.

GNU/Linux: sistema operacional GNU 100% livre que utiliza o Linux como kernel. GNU/Linux é a variante mais conhecida do sistema GNU.

GPL: licença de software livre mais utilizada no mundo. A característica é que qualquer trabalho derivado de um software livre sob GPL, obrigatoriamente deve permanecer livre. A GPL pode ser obtida em <http://www.gnu.org/licenses/gpl.html>, uma cópia é apresentada no apêndice ??.

Código Aberto: expressão utilizada para indicar que você pode ver o código fonte do programa. Entretanto, nada pode ser dito a respeito das condições sob as quais o código fonte se encontra. Existem programas de código aberto que não são livres pois o usuário, dependendo da licença, pode ser proibido de alterar e publicar o código.

Open Source Initiative: grupo desmembrado do movimento pelo software livre em 1998 e que rejeita a luta pela liberdade no uso do software. Seu objetivo é disseminar a idéia de que o código fonte esteja disponível e que as modificações sejam permitidas gerando assim programas melhores. Apesar disto este grupo costuma recomendar o uso da licença GPL, que representa melhor o espírito do software livre. Isto torna os dois grupos cooperativos do ponto de vista de criação de softwares.

Software Livre: Software que garante ao usuário plenas liberdades de uso, modificação, cópia e publicação de versões modificadas. A definição de software livre pode ser encontrada em: <http://www.gnu.org/philosophy/free-sw.html>.

Movimento pelo Software Livre: Movimento surgido na década de 70 em consequência da crescente pressão recebida para a adoção de softwares proprietários e assinaturas de tratados de não-divulgação. O movimento ganhou força a partir da década de 80 com o projeto GNU que libertava os usuários dos sistemas UNIX proprietários. O GNU consolidou-se na década de 90 como um sistema completo e funcional atingindo uma qualidade técnica comparável aos melhores sistemas operacionais proprietários.

Em resumo, Linux é um kernel e não um sistema operacional. Apesar de “Linux” ser comumente usado em referência ao sistema operacional GNU/Linux você deve evitar este uso equivocado por

²Contribuição do revisor Ricardo Grutzmacher (grutz@terra.com.br).

questões de clareza técnica e por questões de crédito ao projeto GNU que forneceu o seu sistema operacional para ser adaptado ao kernel Linux. Sempre que você quiser falar do sistema operacional use o termo GNU/Linux. Desta forma você estará levando adiante os ideais do software livre que estão representados no projeto GNU.

1.2 O básico do GNU/Linux, Unix

Apresenta-se a seguir um guia com alguns comandos de shell do GNU/Linux, Unix, MacOS X, e que são úteis para programadores.

1.2.1 Comandos do shell úteis³

Lista-se a seguir alguns comandos de shell úteis para programadores. Uma descrição mais detalhada dos mesmos pode ser encontrada em apostilas/livros sobre o GNU/Linux, Unix. Você pode obter informações simplificadas sobre estes comandos usando o programa `man` (de manual). Use o comando **`man nomeDoPrograma`** para ver um manual simples do programa. Versões mais atualizadas dos manuais destes programas são obtidos com **`info nomeDoPrograma`**. Você também pode obter um help resumido usando **`nomeDoPrograma - -help`**.

Nos exemplos apresentados a seguir, os comentários após o sinal `#` não devem ser digitados. Se você já é um usuário experiente do GNU/Linux pode pular esta parte.

Diretórios

- . Diretório atual.
- .. Diretório pai (Ex: `cd ..`).
- ~ Diretório HOME do usuário (Ex: `cd ~`).

cd Muda diretório corrente

```
cd /home/philippi #Caminho completo
cd ../../usr      #Caminho relativo
cd -              #Alterna para diretório anterior
```

pwd Mostra a path do diretório corrente.

ls Lista o conteúdo do diretório.

- | | |
|----|------------------------------|
| -l | Lista detalhada. |
| -a | Mostra executável e ocultos. |
| -b | Número de links do arquivo. |
| -m | Mostra tudo em uma linha. |
| -F | Mostra \ dos diretórios. |
| -x | Mostra em colunas. |

³Uma excelente apostila de referência sobre o GNU/Linux é encontrada em <http://www.cipsga.org.br>.

```
ls -F | egrep /      #mostra diretórios
```

tree Lista em árvore.

```
tree -d              #Lista somente os diretórios
```

mkdir Cria diretório.

```
mkdir test
mkdir d1 d2          #Cria diretórios d1 e d2
mkdir d1/d2          #Cria d1 e o filho d2
mkdir -p d3/d31/d32 #Cria os diretórios d3 d3/d31 d3/d31/d32
```

mvdir Move ou renomeia um diretório.

rmdir Remove o diretório.

```
-R          #Recursivo, elimina subdiretórios (usar com cuidado).

rmdir -p d3/d31/d32 #Remove todos os diretórios
rm -R diretorio     #Remove o diretório e seus sub-diretórios
```

Arquivos

cp a1 a2 Cópia arquivos e diretórios.

```
-b          Cria backup de a2.
-i          Cópia iterativa.
-r          Cópia recursiva.
-P          Cópia arquivo e estrutura de diretório.
-p          Preserva as permissões e horas.
-v          Modo verbose.
-b          Cria backup.
```

```
cp a1 a2
```

mv Move arquivos (renomeia arquivos).

```
-b          Cria backup.
-v          Modo verbose.
-i          Iterativa.
```

```
mv a1 a2          #Renomeia arq a1 para a2
mv d1 d2          #Renomeia dir d1 para d2
mv -b a1 a2       #Renomeia com backup
```

rm Remove arquivos (retira links).

- d Remove diretório.
- i Remove iterativamente.
- r Remove diretórios recursivamente.
- f Desconsidera confirmação.

```
#Só execute o comando abaixo em um subdiretório sem importância
rm -f -r * #Remove tudo (*) sem pedir confirmação
```

ln Linka arquivos e diretórios (um arquivo com link só é deletado se for o último link).

```
ln -f chap1 intro
ln origem link #Cria link permanente
ln -s origem link #Cria link simbólico
```

find O find é usado para pesquisar arquivos em seu HD.

- find path expressão*
- name *Informa o nome do arquivo.*
- print *Mostra a path.*
- type *Informa o tipo.*
- atime *Informa dados de data.*
- size *Informa tamanho(+ ou -).*
- o *Aceita repetição de parâmetro (Other).*
- xdev *Desconsidera arquivos NFS.*
- exec [comando [opcoes]] *Permite executar comando.*

```
#Para achar arquivos core:
find / -name core
#Para localizar arquivos do usuário:
find PATH -USER nomeUsuário
#Para localizar arquivos *.cpp:
find PATH -name *.cpp
#Para localizar e remover arquivos .o:
find PATH -name *.o | xargs rm
#Localizando tudo com a extensão *.o e *.a
find -name *.o -o -name *.a
#PS: exec só funciona com alguns programas, melhor usar xargs.
find -type f -atime +100 -print
find . -size +1000k
find ~/ -size -100k
find [a-c]????
find file[12]
```

head n Mostrar as primeiras n linhas de um arquivo.

```
head -5 nome.txt
```

tail n Exibe arquivo a partir de uma linha.

```
tail -20 nome.txt
```

nl Número de linhas do arquivo.

wc Número de linhas, de palavras e de bytes do arquivo.

cat arq1 Mostra conteúdo do arquivo arq1.

cat a1 a2 Mostra arquivos a1 e depois a2

cat -n a2 Mostra conteúdo de a2 com numeração

```
cat f1                                #Mostra arquivo f1
                                     #Cria novo arquivo:
cat > texto.txt
...digita o texto...
ctrl+d                               #Finaliza arquivo
cat a1 a2 > a3                       #Concatena a1 e a2 e armazena em a3
cat a >> b                           #Acrescenta ao final do arquivo b o arquivo a
ls -la | cat -n
```

less arq Mostra conteúdo do arquivo (+completo)

```
#/str para localizar a string str no texto visualizado
less arq #q para sair
```

file arq Informa o tipo de arquivo.

```
file *
file * | egrep directory
```

tr Converte cadeias de caracteres em arquivos.

```
ls | tr a-z A-Z #de minúsculas para maiúsculas
```

xargs Facilita passagem de parâmetros para outro comando.

```
xargs [opções][comando [opções]]
```

```
#Procura e deleta arquivos *.cpp
```

```
find /tmp -name "*.cpp" | xargs rm
```

Pesquisa dentro de arquivos

grep O grep é usado para pesquisar o que tem dentro de um arquivo.

- n Mostra número da linha.
- F O grep funciona como o fgrep.
- c Retorna número de coincidências.
- i Desconsidera maiúsculas/minúsculas.
- s Desconsidera mensagens de erro.
- v Modo verbose.
- A n Lista também as n linhas posteriores.
- B n Lista também as n linhas anteriores.
- r Ordem inversa.
- f Usa arquivo auxiliar.

```
man grep           #Mostra detalhes do grep
grep -v bubble sort.c
ls -l | egrep "Julia" #Lista diretório e pesquisa pelo arquivo julia
ls -la | egrep "Sofia"
grep ^[0-9] guide.txt
grep "(b)" guide.txt
grep arqAux guide.txt # Pesquisa em guide.txt usando arqAux
```

sort Ordena arquivos.

- c Verifica arquivo.
- o Especifica nome arquivo saída.
- d Ordem dicionário.
- f Despreza diferença maiúscula/minúscula.
- t Atualiza data e hora.
- s Modo silencioso.

```
sort -r arquivo
ls | sort -r           #Ordem invertida
#Ordena a listagem de diretório a partir da 4 coluna, considerando número
ls -l | egrep rwx | sort +4n
```

Compactação e backup

zip Compatível com pkzip/pkunzip do DOS.

unzip Unzipa arquivos zip.

```
zip -r nome.zip nomeDiretório
unzip nome.zip
```

gzip / gunzip Compacta/Descompacta arquivos com a extensão: gz,.Z,-gz,.z,-z

- c Mostra arquivo na tela.
- d Descomprime o arquivo.
- S Extensão do arquivo.
- f Força compressão.
- l Lista arquivos.
- r Mostra diretórios recursivamente.
- t Testa integridade do arquivo.
- v Modo verbose.
- 1 Mais veloz e menos compactado.
- 9 Mais lento e mais compactado.

```
#Para compactar todo um diretório
tar -cvf nomeDiretorio.tar nomeDiretorio
gzip nomeDiretorio.tar
#Para descompactar
gunzip nomeDiretorio.tar.gz
tar -xvf nomeDiretorio.tar
```

bzip2 Compactador mais eficiente que o gzip.

bunzip2 Descompacta arquivos bz2.

bzip2recover Recupera arquivos bz2 extragados.

- t Testa
- v Modo verbose

bz2cat Descompacta para tela (stdout).

lha Cria e expande arquivos lharc.

unarj Descompacta arquivos arj.

split Útil para copiar um arquivo grande para disquetes.
Gera os arquivos xaa,xab,xac,... Veja man split.

```
#Dividir um arquivo de 10mb em disketes de 1.44mb:
split -b 1440kb nomeArquivoGrande.doc
#Para recuperar use o cat:
cat xaa xab xac > nomeArquivoGrande.doc
```

tar O tar permite a você agrupar um conjunto de arquivos em um único, facilitando o backup (ou o envio pela internet). Pode-se ainda compactar os arquivos agrupados com o gzip.

-c	Cria novo arquivo.
-v	Modo verbose.
-z	Descompacta arquivos (*.tar.gz).
-f	NomeArquivo.
-w	Modo iterativo.
-r	Acrescenta no modo apende.
-x	Extraí arquivos.
-A	Concatena arquivos.
-d	Compara diferenças entre arquivos.
- -delete	Deleta arquivos.
-t	Lista o conteúdo do arquivo.
-u	Atualiza o arquivo.
-N	Após a data tal.
-o	Extraí arquivos para monitor.
-w	Iterativa.
-C	Muda o diretório.
-G	Backup incremental.

```
#Empacotando origem em destino
tar -cf origem > destino.tar
#Empacotando e já compactando
tar -cvzf destino.tar.gz origem
#Desempacotando
tar -xf nome.tar
tar -xzvf nome.tgz
tar -xzvf nome.tar.gz
tar -xvzf nome.tar
#Backup completo no dispositivo /dev/fd0
tar cvfzM /dev/fd0 /          #Backup
tar xvfzM /dev/fd0           #Restauração
```

Diversos

[*] Metacaracter.

[?] Caracter coringa.

clear Limpa a tela.

date Mostra data e hora.

```
date -s "8:50"      #Acerta hora para 8:50
```

1.2.2 Expressões regulares³

Alguns programas do GNU/Linux aceitam o uso de expressões regulares (principalmente os da GNU). O uso de expressões regulares é útil nas pesquisas de arquivos com find, no uso do emacs, entre outros.

Alguns caracteres são usados para passar parâmetros para o interpretador das expressões regulares, para usá-los na pesquisa, deve-se preceder o caracter com /.

Veja abaixo alguns exemplos de expressões regulares.

[xyz]	Qualquer das letras dentro do colchete.
[^xy]	Exceto x e y.
[t-z]	De t até z (tuvwxyz).
[a-zA-Z]	Qualquer letra.
[0-9]	Qualquer número.
^	No início do parágrafo.
\$	No fim do parágrafo.
\<	No início da palavra.
\<search	Palavra que inicia com search.
\>	No fim da palavra.
\>search	Palavra que termina com search.
z*	Letra z, 0 ou mais vezes: z,zz,zzz,zzzz,...
Z+	Letra Z, 1 ou mais vezes.
A?	Letra A pode ou não fazer parte.
revistas?	revista ou revistas.
A{m}	Letra A m vezes.
[0-9]{3}	Número de 3 dígitos.
Z{3,5}	Letra Z de 3 a 5 repetições zzz,zzzz,zzzzz
K{,7}	Até 7 repetições de K.
K{3,}	No mínimo 3 repetições de K.
{0,}	O mesmo que *.
{1,}	O mesmo que +.
{0,1}	O mesmo que ?.
()	Usado para deixar a expressão mais clara (precedências).

Linux-(6|6.1|6.2), Acha Linux-6, Linux-6.1, Linux-6.2.

O uso de () cria variáveis internas que podem ser acessadas como nos exemplos abaixo:

(quero)-\1 = quero-quero.

[a-zA-Z]\1 Qualquer letra espaço qualquer letra.

. Qualquer caracter. Se estiver no meio de uma sentença use \.

\w Qualquer letra.

\W Nenhuma letra.

| Pipe (tubo).

Conversão de wildcard (dos), para expressão regular.

*.txt *\txt.

arq.cpp Arq\cpp.

arq?.cpp Arq.\cpp.

Cap[1-7].lyx Cap[1-7]\lyx.

arq{a,b} arq(a|b).

Exemplo:

var=Avancado Define variável var, cujo conteúdo é o diretório Avancado.

Para usar uma variável definida use \$var.

ls var Não aceita (ls: var: Arquivo ou diretório não encontrado).

ls \$var Mostra o conteúdo de var, do diretório Avancado.

ls 'var' Não aceita aspas simples.

ls "echo \$var" Não interpreta conteúdo de aspas duplas.

ls 'echo \$var' Interpreta o conteúdo que esta dentro da crase.

for i in *.html; do 'mv \$i \$i.old' ; done

1.2.3 Programas telnet e ftp

Apresenta-se a seguir os principais comandos dos programas telnet e ftp. Atualmente o telnet e o ftp estão sendo substituídos pelo ssh. O ssh é um secure shell, permitindo acessar, copiar e pegar arquivos de máquinas remotas. Isto é, o ssh funciona como o telnet e o ftp, mas com segurança.

telnet

O telnet é um programa que é executado em seu computador e o conecta a outros computadores em qualquer lugar do mundo. É como se você estivesse executando os programas do computador remoto de dentro de seu computador.

Comando	Descrição.
?	Lista os comandos.
open	Conecta com um computador remoto.
display	Mostra os parâmetros da operação.
mode	Modo linha a linha ou caracter a caracter.
Set,unset	Seta os parâmetros de operação.
send	Transmitir caracteres especiais.
status	Informações de estatus.
Contrl +z	Suspende o telnet, chama o shell.
fg	Retorna do shell para o telnet. Volta para o último programa em operação.
logout	Encerra conexão avisando.
close	Fecha a conexão corrente.
quit	Encerra o telnet.

ftp

O ftp precisa de 2 computadores, um cliente e outro servidor. O programa ftp cliente é executado em seu computador e o programa ftp servidor em um computador remoto. O cliente envia comandos (como listar diretórios) para o servidor, o servidor responde (mostrando o diretório).

Depois de localizado o arquivo (que você deseja baixar), o mesmo pode ser obtido com o comando `get nome_do_arquivo`. Com `quit` você encerra a conexão ftp.

help	Lista de todos os comandos ftp.
!	Pausa o ftp e inicia o shell.
! comando	Executa um comando do shell.

Conectando

open h	Inicia conexão com host h.
user	Define usuário.

ascii	Transferência de arquivo no formato ascii.
binary	Transferência de arquivo no formato binário.
hash yes/no	Mostra o # do arquivo transferido.
prompt yes/no	Aguarda resposta para transferência de múltiplos arquivos.
status	Exibe configuração atual.
get	Puxa o arquivo para seu computador.
mget	Puxa diversos arquivos.
send	Envia um arquivo (ou put, ou mput).
cd nome	Muda o diretório.
cdup	Diretório pai.
dir ou ls	Mostra diretório com detalhes.
lcd	Muda diretório local.
pwd	Mostra a path corrente.
quit	Encerra a conexão.
close	Encerra conexão.

Agora você já conhece os comandos e programas mais utilizados em um terminal do GNU/Linux. Podendo navegar, criar, deletar e mover diretórios. Procurar e visualizar o conteúdo de arquivos.

Como dito anteriormente, você pode consultar o manual de sua distribuição para obter maiores informações sobre estes comandos.

1.3 Diretórios úteis para programadores

Quem programa no GNU/Linux precisa saber onde estão o diretório com os programas do sistema, o diretório onde são instalados os programas e onde estão as bibliotecas. Veja na Tabela ?? alguns destes diretórios.

Tabela 1.1: Diretórios importantes para o programador.

Diretório	Descrição
/usr/bin	Programas do sistema.
/usr/local/bin	Programas locais estão aqui.
/usr/include	Arquivos include
/usr/lib	Bibliotecas
/usr/openwin/lib	Bibliotecas do X window

1.4 Programas úteis para programadores

Apresenta-se na Tabela ?? uma lista de programas úteis usados no desenvolvimento de programas no GNU/Linux. Estes programas serão detalhados posteriormente.

Tabela 1.2: Programas úteis para desenvolvedores de software no ambiente GNU/Linux.

	Programas utilitários	
	Ambientes de desenvolvimento	
	Ambiente para desenvolvimento no KDE	kdevelop
	Ambiente de desenvolvimento semelhante ao Borland	xwpe
	Ambiente para desenvolvimento no GNOME	glade
	Editor de texto	
	Editor de texto simples	emacs
	Compiladores	
	Compilador C da GNU	gcc
	Compilador C++ da GNU	g++
	Compilação automatizada	make
	Linkagem	ld
	Depuração	
	Depurador C da GNU	gdb
	Depurador do KDE (kdevelop)	kdbg
	Programas GNU Compliant	
	Geração dos scripts de configuração	autoconf
	Geração dos scripts Makefile	automake
	Pacote de geração de bibliotecas	libtool
	Programas Diversos	
	Traça informações, chamadas a bibliotecas	ltrace
	Controle de versões	CVS
	Formatação do código	
	Diferenças entre arquivos	diff
	Formata o código	bison
	Organiza o código (layout)	indent
	Analisador Léxico	flex, flex++
	Documentação	
	Gera documentação a partir de tags no código.	doxygen
	Geração de diagramas de classes	graphviz
	Manipulação binária	bfd
	Binary file descriptor	binutil
	Profile (analisador de desempenho)	gprof
	Montagem de bibliotecas	ar
	Índices das bibliotecas	ranlib
	Informações sobre objetos	objdump

Capítulo 2

Edição de Texto Emacs e VI

Apresenta-se neste capítulo uma breve descrição dos comandos dos editores emacs e vi.

2.1 Introdução ao editor emacs

O emacs, ilustrado na Figura ?? é um bom editor de texto. O emacs oferece um conjunto de funções específicas para o desenvolvimento de seus programas em C++.

Ao salvar o arquivo com a extensão *.h ou *.cpp, automaticamente o emacs mostra a sintaxe de C++ através do “syntax-highlight”, isto é, o texto aparece colorido.

Para maiores informações sobre o emacs procure no <http://www.altavista.com> por *emacs programming*.

Se você gosta de outros editores de texto, não tem problema. Use o que você conhece.

Os principais comandos de um editor de texto são os famosos cut (ctrl+w), copy (alt+w) e past (ctrl+y). O undo (ctrl+x+u), o search (ctrl+s+palavra), e o comando para salvar o arquivo (ctrl+x ctrl+s) e fechar o programa (ctrl+x ctrl+c). Apresenta-se nas seções que seguem os principais comandos do editor emacs.

2.2 Comandos do editor emacs

Lista-se a seguir, de forma resumida os principais comandos do emacs. A letra ^ representa o CTRL.

2.2.1 Help

^+h n	Emacs news.
^h i	Info.
^h m	Modo de descrição.
^h a	Comando apropos.
^h t	Emacs tutorial.
^h f	Descrição da função.

`^x` Acesso a função de um único caracter.

`Alt-x` Acesso a comando de uma palavra.

2.2.2 Movimento do cursor (use as setas de direção)

`Alt+<` Início do arquivo.

`Home` Início do arquivo.

`pageUp` Início da página.

`^p` Início do parágrafo.

`^b ^f` Linhas.

`^a ^e` Fim do parágrafo.

`Alt+b Alt+f` Palavras.

`Alt+>` Fim do arquivo.

`pageDown` Fim da página.

`End` Fim do arquivo.

2.2.3 Cut/Copy/Paste/Undo

`Alt+w` Copy.

`^w` Cut.

`^y` Paste.

`^x u` Undo.

`^_` Undo.

`Alt+y` Paste sucessivo.

`^d` Apaga a letra a direita.

`del` Apaga a letra a esquerda.

`Alt+d` Apaga a palavra a direita.

`Alt+del` Apaga a palavra a esquerda.

`^k` Apaga toda a linha a direita.

`Alt+k` Apaga toda a linha a direita inclusive retorno de carro.

2.2.4 Arquivos

<code>^x ^f</code>	Abre um arquivo do disco ou cria novo.
<code>^x ^d</code>	Abre o diretório.
<code>^x ^s</code>	Salva o arquivo em disco.
<code>^x ^w</code>	Salvar como.
<code>^x ^d</code>	Abre um diretório.
<code>^x 1</code>	Uma janela.
<code>^x 2</code>	Duas Janelas.
<code>^x i</code>	Inserir o arquivo.
<code>^x ^b</code>	Lista os buffers.

2.2.5 Pesquisa e substituição

<code>Alt+%</code>	Entre com a string A Entre com a string B #Pede confirmação
<code>Alt+x repl s</code>	Entre com a string A Entre com a string B #Não pede confirmação

Pesquisa

<code>^s palavra</code>	Procura pela palavra (para baixo).
<code>^r palavra</code>	Procura pela palavra (para cima).

2.2.6 Múltiplas janelas

<code>^u 0^]</code>	Posiciona no início da janela
<code>^mv</code>	Scroll para final da janela
<code>^xo</code>	Alterna janela ativa
<code>^x2</code>	Duas janelas
<code>^x1</code>	Uma janela ativa

2.2.7 Encerrando seção do Emacs

<code>^x ^c</code>	Sai do Emacs.
<code>^z</code>	Suspende o Emacs.

Sentenças para o emacs

- AUTO SAVE: O emacs salva automaticamente os arquivos em disco com o nome: "#nomeArquivo#". Quando você salva o arquivo em disco deixa de existir o autoSave.
- A Linha de informações mostra: O nome do arquivo, a linha, a percentagem

2.3 Comandos do editor vi

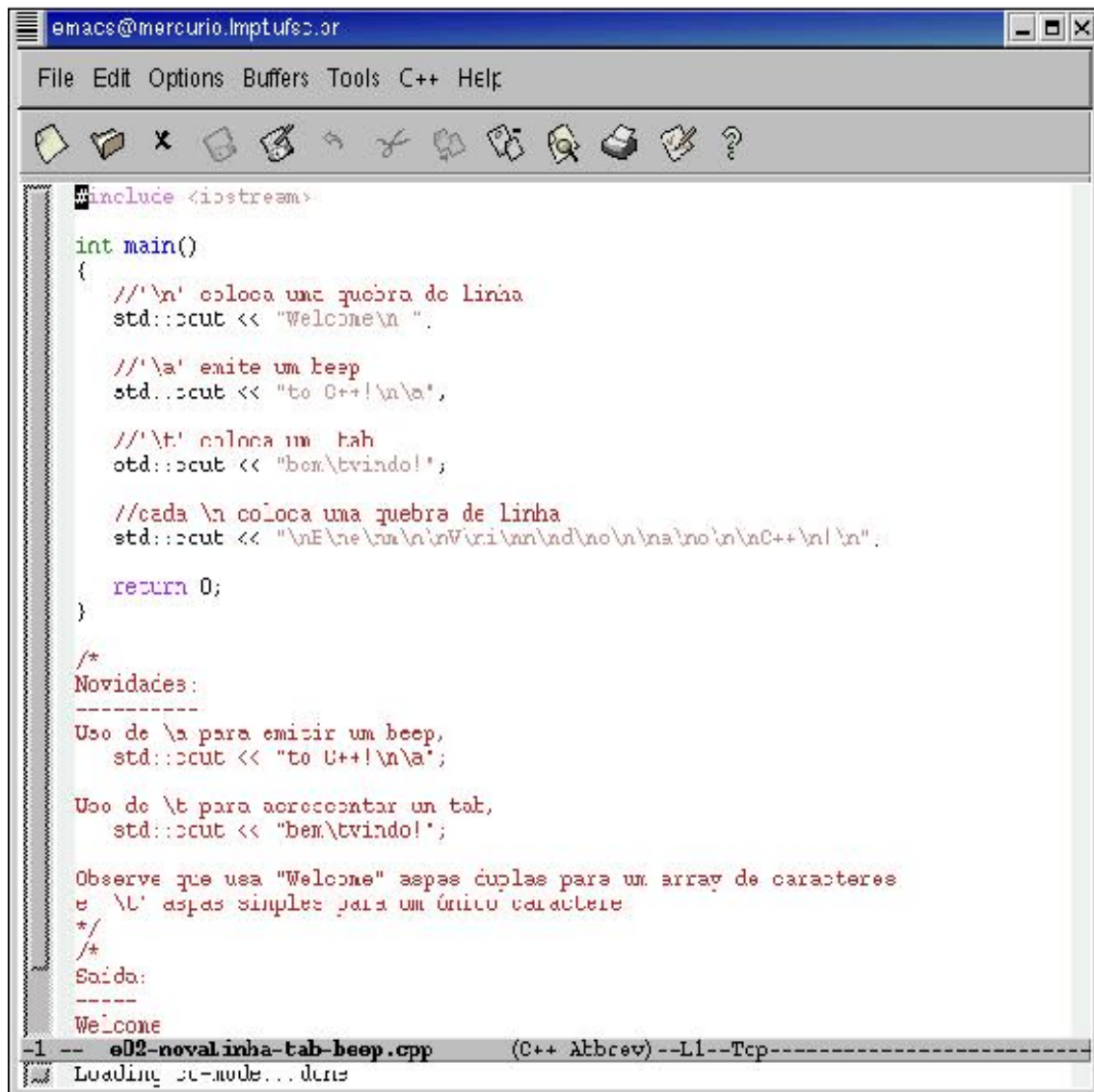
Editor de texto simples e eficiente.

:w	Salva arquivo em disco.
:q	Para sair.
:q!	Sai mesmo que o arquivo tenha sido alterado.
:e	Edita outro arquivo.
:!com	Executa comando do shell.
:r arq	Lê outro arquivo na posição do cursor.

Exemplo:

```
vi arquivo #Edita o arquivo
man vi     #Maiores informações sobre o vi
```

Figura 2.1: O editor de texto emacs (observe o item de menu C++).



Capítulo 3

Os programas diff, patch, indent

Apresenta-se neste capítulo uma breve introdução aos programas diff, patch e indent. Este capítulo pode ser lido mais tarde sem perda de seqüência.

3.1 O programa diff

O programa diff é usado para mostrar as diferenças entre 2 arquivos, isto é, compara dois arquivos linha a linha.

O programa diff é muito útil, o mesmo é usado para comparar versões de um mesmo arquivo (veja Capítulo ??) e para gerar arquivos de patch (veja seção ??).

Protótipo e parâmetros do diff:

diff [opções] Arq1 Arq2

- b* Ignora espaços em branco.
- c* Gera saída mostrando os 2 arquivos e as diferenças.
- i* Ignora diferenças entre maiúsculas e minúsculas.
- q* Apenas informa se os arquivos são iguais ou diferentes.
- r* Compara diretórios recursivamente.
- v* Mostra versão do diff.
- xpattern* Quando compara diretórios, considerar arquivos com a extensão pattern.
- u* Formato unificado (+ claro).

Veja a seguir o arquivo e06a-hello.cpp, um programa simples em C++.

Listing 3.1: Arquivo e06a-hello.cpp.

```
#include <iostream>

int main()
{
    std::cout << "Oi_tudo_bem_"<<std::endl;
    return 0;
}
```

O arquivo e06a-hello.cpp foi modificado e salvo como e06b-hello.cpp. Veja a seguir o arquivo e06b-hello.cpp, observe que foram acrescentadas 4 linhas novas e o return foi alinhado.

Listing 3.2: Arquivo e06b-hello.cpp.

```
#include <iostream>

int main()
{
    std::cout << "Oi_tudo_bem_"<<std::endl;
    std::cout << "Entre_com_x_"<<std::endl;
    int x;
    cin>>x;
    std::cout << "x=_"<<x<<std::endl;

    return 0;
}
```

Veja a seguir o arquivo gerado pelo comando: **diff e06a-hello.cpp e06b-hello.cpp**. O símbolo < indica que esta linha saiu (é velha). O símbolo > indica que esta linha entrou (é nova).

Listing 3.3: Arquivo diff.

```
6c6,11
<     return 0;
---
>     std::cout << "Entre com x "<<std::endl;
>     int x;
>     cin>>x;
>     std::cout << "x= "<<x<<std::endl;
>
>     return 0;
```

Observe a linha 6c6,11. A mesma significa que a linha 6 do arquivo e06a-hello.cpp e a linha 6 do arquivo e06b-hello.cpp são diferentes, existe um espaço extra no arquivo e06a-hello.cpp. O **c** indica modificado (changed). Podem aparecer as letras **a** de adicionado, **c** de modificado (changed) e **d** de deletado.

Veja a seguir o arquivo gerado pelo comando: **diff -c e06a-hello.cpp e06b-hello.cpp**. Observe que os dois arquivos são mostrados na íntegra. O caracter **!** indica as linhas que foram alteradas.

Listing 3.4: Arquivo diff -c.

```
*** e06a-hello.cpp      Tue Jun  4 13:23:49 2002
--- e06b-hello.cpp      Tue Jun  4 13:24:46 2002
*****
*** 3,8 ****
    int main()
    {
        std::cout << "Oi tudo bem "<<std::endl;
!       return 0;
    }

--- 3,13 ----
    int main()
    {
        std::cout << "Oi tudo bem "<<std::endl;
!       std::cout << "Entre com x "<<std::endl;
```

```
!   int x;
!   cin>>x;
!   std::cout << "x= " <<x<<std::endl;
!
!   return 0;
}
```

Veja a seguir o arquivo gerado pelo comando: **diff -u e06a-hello.cpp e06b-hello.cpp**. No início do arquivo a nomenclatura - - - se refere ao arquivo **e06a-hello.cpp** e +++ ao arquivo **e06b-hello.cpp**.

Observe que no formato -u aparecem todas as linhas precedidas por um sinal +/- . O sinal - indica que a linha saiu e o sinal + que entrou.

Listing 3.5: Arquivo diff -u.

```
--- e06a-hello.cpp      Tue Jun  4 13:23:49 2002
+++ e06b-hello.cpp      Tue Jun  4 13:24:46 2002
@@ -3,6 +3,11 @@
 int main()
 {
     std::cout << "Oi tudo bem " <<std::endl;
-   return 0;
+   std::cout << "Entre com x " <<std::endl;
+   int x;
+   cin>>x;
+   std::cout << "x= " <<x<<std::endl;
+
+   return 0;
}
```

3.1.1 Sentenças para o diff

- O formato gerado pelo diff -u é o mais claro.
- Você pode gerar um arquivo com as diferenças.

Exemplo:

```
diff e06a-hello.cpp e06b-hello.cpp > diferencas.txt
```

- O formato padrão gerado pelo diff é usado pelo programa patch.
- Existem outros programas para comparar arquivos, veja nas páginas man de seu GNU/Linux os programas: diff3 (compara 3 arquivos), wdiff, mp, sdiff.
- Ao editar a saída do comando diff no programa emacs, o mesmo aparece com sintaxe especial.

3.2 O programa patch¹

O programa path é usado para unificar arquivos.

¹Veja na seção ?? o uso dos programas diff e patch para distribuição de upgrades de programas.

Protótipo e parâmetros do patch:

patch [-u-v] arquivoAntigo arquivoDiferencas.

-u O arquivo das diferenças foi gerado usando a versão unificada (opção *-u* no *diff*).

-v Mostra versão do programa *patch*.

Dados os arquivos *arq1.cpp*, *arq2.cpp* e o arquivo das diferenças gerado pelo *diff*, você pode atualizar o arquivo *arq1.cpp* de forma que o mesmo tenha o mesmo conteúdo do *arq2.cpp*. Veja o exemplo.

Exemplo:

```
diff arq1.cpp arq2.cpp > diferencas.cpp
//Vai modificar o arq1 e o mesmo ficará igual a arq2
patch arq1.cpp diferencas.cpp
```

Ou seja, o programa *patch* junta os arquivos *arq1.cpp* e *diferencas.cpp* criando o arquivo *arq2.cpp*.

3.3 O programa indent

O programa *indent* é usado para deixar o código organizado, através do uso de padrões de indentação. Existe um padrão default, pré-estabelecido, mas você pode definir os padrões a serem utilizados. Veja a seguir o protótipo de uso do *indent*. Lista-se, de forma abreviada, algumas das opções que podem ser passadas para o *indent*. Estes parâmetros podem ser armazenados no arquivo “.indent.pro”.

Protótipo:

indent file [-o outfile] [opções]
indent file1 file2 ... fileN [opções]

-st Envia saída para tela.

-gnu Usa opções de formatação da GNU.

-orig Usa opções de formatação da Berkeley.

-v Modo verbose.

-l60 Limita a 60 colunas.

-bad Linha em branco após declarações (Para desativar, *-nbad*).

-bap Linha em branco após definições de funções (*-nbap*).

-bbb Linha em branco antes de uma caixa de comentário.

-sc Comentários no estilo de C /* * */.

-bl5 Colchetes do bloco alinhados a 5 caracteres.

-bn Bloco alinhado.

-bli5 Bloco alinhado com 5 espaços.

-bls Alinha o par{ }.

-cli2 Bloco switch alinhado com espaço 2.

<i>-npcs</i>	<i>Sem espaço entre o nome da função e o ().</i>
<i>-cs</i>	<i>Espaço depois do cast.</i>
<i>-di16</i>	<i>Indenta nome dos objetos em 16 espaços.</i>
<i>-bfda</i>	<i>Quebra argumentos da função em várias linhas.</i>
<i>-lp</i>	<i>Alinha parâmetros de funções com nomes grandes.</i>

O programa *ex-vector-1.cpp* apresentado na seção ??, foi modificado com o programa indent com o comando:

```
Exemplo:
cp ex-vector-1.cpp ex-vector-1-indent.cpp
indent ex-vector-1-indent.cpp
```

veja a seguir a listagem do arquivo *ex-vector-1-indent.cpp*. Compare esta listagem com a listagem ??. Observe a mudança na declaração da função *main*, na forma do *do..while* e nas indentações.

Listing 3.6: Arquivo *ex-vector-1-indent.cpp*.

```
//Classes para entrada e saída
#include <iostream>

//Classe pra formatação de entrada e saída
#include <iomanip>

//Classe de vetores, do container vector
#include <vector>

//Classe para algoritmos genéricos
//#include <algorithm>

//Define estar usando espaço de nomes std
using namespace std;

//Definição da função main
int
main ()
{
    //Cria vector, do tipo int, com nome v
    vector < int >v;

    int data;
    cout << "No_DOS_umlctrl+d_encerra_a_entrada_de_dados." << endl;
    cout << "No_Mac_umlctrl+d_encerra_a_entrada_de_dados." << endl;
    cout << "No_Linux_umlctrl+d_encerra_a_entrada_de_dados." << endl;
    do
    {
        cout << "\nEntre_com_o_dado_(" << setw (3) << v.size () << "):";
        cin >> data;
        cin.get ();
        //acidiona ao final do vetor o objeto data
        v.push_back (data);
    }
}
```

```

while (cin.good ());

//Acessa partes do vetor usando funções front e back
cout << "\nPrimeiro elemento do vetor=" << v.front ()
    << "\nÚltimo elemento do vetor=" << v.back () << endl;

//mostra o vetor
for (int i = 0; i < v.size (); i++)
{
    cout << "v[" << setw (3) << i << "]= " << setw (5) << v[i] << ' ';
}
cout << endl;

cout << (v.empty ()? "vazio" : "não_vazio") << endl;

//Chama função clear
v.clear ();
cout << (v.empty ()? "vazio" : "não_vazio") << endl;

cout << endl;
cin.get ();
return 0;
}

```

Sentenças para o indent

- Opções padrões do padrão -gnut
 -nbad -bap -nbc -bbo -bl -bli2 -bls -ncdb -nce -cp1 -cs -di2 -ndj -nfc1 -nfca -hnl -i2 -ip5 -lp -pcs
 -nprs -psl -saf -sai -saw -nsc -nsob
- No código você pode desabilitar ou habilitar o indent.
 /* * indent_on */,
 /* * indent_off */.
- Para maiores informações sobre o indent consulte as informações do programa usando **info indent**.
- Para ver todas as opções do indent use: **man indent**.

Capítulo 4

Compilando com gcc, g++

Neste capítulo apresenta-se o compilador gcc/g++ da GNU. O mesmo pode ser baixado no site da gnu individualmente, ou como parte do pacote do EGCS (que inclui um conjunto de programas acessórios).

4.1 Protótipo e parâmetros do gcc/g++

Apresenta-se aqui o protótipo e os parâmetros que você pode passar para o compilador da gnu. Posteriormente, através dos exemplos, você compreenderá melhor cada parâmetro.

Protótipo do gcc/g++:

g++ [opções] [parâmetros] arquivos.

- v*** *Mostra detalhes da compilação.*
- wall*** *Mostra todos os warnings.*
- onome*** *Define o nome do arquivo de saída (opcional).*
- w*** *Elimina mensagens de warning.*
- I/path*** *Acrescenta path include.*
- l/path/lib*** *Inclue biblioteca (lib).*
- ggdb*** *Informações extras para o gdb.*
- O*** *Otimiza o código (-O1,-O2,-O3).*
- c*** *Somente compila (gera o arquivo *.o).*
- S*** *Somente compila o arquivo, não linka.*
- lcomplex*** *Inclue biblioteca dos complexos.*
- lm*** *Inclue biblioteca matemática.*
- E*** *Cria somente o arquivo pré-processado.*
- C*** *Não inclue comentários no executável.*
- g*** *Gera informações para o debugger (código lento).*
- qdigraph*** *Adiciona teclas dígrafas.*

- qcompact* Deixa o código mais compacto.
- xlinguagem* Especifica a linguagem (C, C++, assembler).
- p* Informações para o profiler proff.
- pg* Informações para o groff.
- m686* Especifica que a máquina alvo é um 686.
- static* Especifica que a linkagem deve ser estática.
- p* Especifica inclusão de instruções para o profiler.
- pg* Especifica inclusão de instruções para o profiler da gnu (gprof).

4.2 Arquivos gerados pelo gcc/g++

A medida que os arquivos são compilados e linkados, são gerados alguns arquivos adicionais. Lista-se a seguir os arquivos de entrada (*.h, *.cpp) e os arquivos de saída gerados pelo g++.

- *.h* Arquivos header.
- *.i* Arquivos de pré-processamento para programas em C.
- *.ii* Arquivos de pré-processamento para programas em C++.
- .c, .cc, .C, .c++, .cpp*, Arquivos de fonte.
- .o* Arquivo objeto.
- .s* Arquivo assembler.
- .a* Arquivo de biblioteca estática.
- .sa* Blocos de bibliotecas estáticas linkados ao programa.
- .so.n* Arquivo de biblioteca dinâmica.
- a.out* Arquivo de saída (executável).

4.3 Exemplo de uso do gcc/g++

Apresenta-se a seguir um exemplo básico.

1. Edita o programa hello.cpp com o emacs.
Abra um terminal e execute **emacs hello.cpp**.

```
#include <iostream>
using namespace std;
int main()
{
    cout << "hello!" << endl;
```

```
return; }  
/*No emacs use ctrl+x ctrl+s para salvar o arquivo  
e ctrl+x ctrl+q para sair do emacs*/
```

2. Cria o arquivo de pré-processamento [opcional].

```
g++ -E hello.cpp
```

3. Compila o módulo hello (gera o hello.o).

```
g++ -c hello.cpp
```

4. Linka o programa e gera o executável.

```
g++ -ohello hello.o
```

5. Executa o programa.

```
./hello
```

6. Observe que os passos 2, 3 e 4 podem ser executados usando:

```
g++ -v -ohello hello.cpp
```

O -v mostra um conjunto de informações a respeito dos passos da compilação.

Neste exemplo o nome do programa executável é hello e foi especificado com a opção -ohello. Se você não especificar um nome, o programa executável se chamará **a.out**.

Tarefa: Compare o tamanho dos códigos (dos executáveis) gerados com as opções:

```
g++ hello.cpp          #comum  
g++ -g2 hello.cpp      #com debugger  
g++ -o3 hello.cpp      #com otimização
```

4.4 Sentenças para o compilador da gnu

- Para conhecer em detalhes o gcc baixe o manual do gcc no site da gnu.

Capítulo 5

Make

No capítulo anterior você aprendeu a usar o g++ para compilar um programa, como você pode constatar, é fácil. Mas se seu programa for grande, você vai ter de executar o g++ para cada arquivo *.cpp, o que se transforma em trabalho tedioso.

Para evitar este trabalho, foram desenvolvidos os arquivos de projeto. Neste capítulo vamos descrever o que é, como escrever e como usar arquivos de projeto e o programa make.

5.1 Um arquivo de projeto¹

Um arquivo de projeto permite a compilação de diversos arquivos com uso do programa make.

- Você pode compilar seus programas diretamente (digitando a instrução de compilação), ou usando arquivos makefile.
- Um arquivo makefile é um arquivo de projeto no formato ASCII, que pode ser editado em editores simples, como o emacs.
- Por padrão, um arquivo de projeto makefile tem o nome **makefile** ou **Makefile**.
- Um arquivo de projeto pode ser simples ou complexo, vai depender do tamanho do programa e dos recursos utilizados.

Podemos criar um arquivo Makefile com instruções de compilação. Estas instruções de compilação serão lidas e executadas pelo programa make, automatizando a compilação de programas complexos. Descreve-se a seguir como funciona o programa make.

5.2 Protótipo e parâmetros do make

Apresenta-se a seguir o protótipo e os parâmetros do programa make. Por padrão, o make procura no diretório corrente o arquivo makefile e depois o arquivo Makefile. A opção -f é usada para passar o nome do arquivo makefile.

¹Um arquivo de projeto do Borland C++ e do MFC tem um formato proprietário, que só pode ser lido por estes programas. Um arquivo de projeto do kdevelop é um arquivo ASCII, podendo ser editado em qualquer editor simples.

Protótipo do make:

make [*-f* *arq_makefile*] [*opções*] ... *alvos* ...

- e* Indica variáveis do ambiente que devem prevalecer sobre atribuições feitas no *make*.
- k* Desconsiderar erros.
- n* Apenas lista os comandos, sem executá-los.
- p* Imprime alguns resultados.
- r* Despreza regras intrínsecas.
- t* Atualiza data e hora.
- s* Modo silencioso.
- f* *arq_makefile* Especifica o nome do arquivo *makefile*.

5.3 Formato de um arquivo Makefile

Basicamente, um arquivo *makefile* é dividido em uma parte com definições de variáveis e outra parte com sub-rotinas a serem executadas.

A primeira parte define variáveis a serem utilizadas. As variáveis incluem o nome do compilador, as paths de inclusão de arquivos e bibliotecas e listagens de arquivos a serem processados.

5.3.1 Criando variáveis em um arquivo Makefile

O *make* aceita que você defina variáveis em seu arquivo *makefile* e depois use estas variáveis nas etapas de compilação.

No exemplo a seguir cria uma variável e usa seu conteúdo. Observe que para obter o valor da variável você deve usar um *\$()*. Ao usar *\$(VARIABEL)*, o *make* substitue o nome da variável pelo seu valor.

Exemplo:

```
NOMEVARIABEL = valor
$(NOMEVARIABEL)
```

- Por convenção, as variáveis definidas em um arquivo *makefile* são todas maiúsculas.
- Você pode criar variáveis da forma

```
CC = g++
```

- E a seguir modificar seu valor, acrescentando algo em modo *append*

```
CC += -O2
```

- ²Make usa um conjunto de variáveis com nomes pré-definidos.

AR Especifica o programa de manutenção de arquivos.

CC	Especifica o compilador, default=cc.
CPP	Especifica o pré-processador C++.
RM	Programa de remoção de arquivos, default = rm -f .
CFLAGS	Flags que devem ser passados para o compilador C.
CPPFLAGS	Flags que devem ser passados para o compilador C++.
LDFLAGS	Flags que devem ser passados para o linker.

5.3.2 Criando alvos em um arquivo Makefile

A segunda parte de um arquivo makefile contém alvos a serem executados. O formato padrão para um alvo é dado por:

Protótipo de um alvo:

alvo: Dependências

Instruções a serem executadas

- As instruções a serem executadas iniciam com um tab (e não 8 espaços).
- Geralmente o alvo é o nome de um arquivo, uma biblioteca ou um programa a ser compilado.
- Alvos usuais em arquivos makefile são:

all	Executar todas as dependências.
install	Instalar o programa.
uninstal	Desinstalar o programa.
dist	Gerar uma distribuição no formato .tar.gz.
check	Verificar a consistência da instalação.
clean	Eliminar arquivos temporários (*.obj).

5.4 Exemplo de um arquivo Makefile

Veja na listagem a seguir um exemplo de arquivo makefile.

Listing 5.1: Arquivo makefile.

```
#Toda linha começada com # é uma linha de comentário
#-----
#ParteI: Definição de variáveis
ARQUIVOS=e06a-hello.cpp
OBJETOS=e06a-hello.o
DIRINCLUDE = -I/usr/include/g++ -I/usr/include
DIRETORIOLIB =
PARAMETROSLIB = -lm
COMPILADOR = g++

#-----
```

```
#Parte II: alvos
#all é o alvo, e06a-hello.o e e06a-hello são as dependências

all : e06a-hello.o e06a-hello

#e06a-hello.o é o alvo, $( ARQUIVOS) são as dependências
#e $( COMPILADOR) .. é a instrução
e06a-hello.o : $(ARQUIVOS)
    $(COMPILADOR) -c $(ARQUIVOS)  $(DIRINCLUDE) $(DIRECL) -o e06a-hello.o

e06a-hello : $(OBJETOS)
    $(COMPILADOR) $(OBJETOS) $(DIRINCLUDE) $(DIRETORIOLIB) $(PARAMETROSLIB)
    -o e06a-hello

clean :
    rm -f *.o *.obj
```

Veja na listagem a seguir uma sequência de execução do programa make usando o arquivo makefile listado em ??.

Listing 5.2: Exemplo de uso do programa make.

```
//Limpa os arquivos anteriores
[andre@mercurio Cap-GNU]$ make clean
rm -f *.o *.obj

//Compila o arquivo e06a-hello.o
[andre@mercurio Cap-GNU]$ make e06a-hello.o
g++ -c e06a-hello.cpp -I/usr/include/g++ -I/usr/include -o e06a-hello.o

//Gera o programa executável
[andre@mercurio Cap-GNU]$ make e06a-hello
g++ e06a-hello.cpp -I/usr/include/g++ -I/usr/include -lm -o e06a-hello

//Veja abaixo que o make não compila novamente o que está atualizado
[andre@mercurio Cap-GNU]$ make e06a-hello
make: 'e06a-hello' está atualizado.

//Limpando os arquivos obj
[andre@mercurio Cap-GNU]$ make clean
rm -f *.o *.obj

//Observe abaixo que o alvo e06a-hello chama o alvo e06a-hello.o
[andre@mercurio Cap-GNU]$ make e06a-hello
g++ -c e06a-hello.cpp -I/usr/include/g++ -I/usr/include -o e06a-hello.o
g++ e06a-hello.cpp -I/usr/include/g++ -I/usr/include -lm -o e06a-hello

Dica: Observe a forma como os alvos foram traduzidos.
O alvo:
e06a-hello : $(ARQUIVOS) $(OBJETOS)
    $(COMPILADOR) $(ARQUIVOS) $(DIRETORIOINCLUDE) $(DIRETORIOLIB) $(
    PARAMETROSLIB) -o e06a-hello

Foi traduzido da forma:
g++ e06a-hello.cpp -I/usr/include/g++ -I/usr/include -lm -o e06a-hello
```

5.5 Sentenças para o make

- Para conhecer em detalhes o make baixe o manual do make no site da gnu (<http://www.gnu.org>).
- Os arquivos especificados nas dependências devem existir. Se não existirem vai acusar erro.
- Os arquivos de cabeçalho *.h também devem ser incluídos nas dependências. Isto evita a mensagem de erro do compilador pela falta dos mesmos.
- Make é inteligente, se você pediu para executar o alvo 2 e este depende do alvo 1, o make executa primeiro o alvo 1 e depois o alvo 2.
- Se alguma dependência sofre modificações, o make recompila os arquivos que foram modificados.
- Alvos sem dependência não são automaticamente executados.
- Ao editar um arquivo makefile ou Makefile no emacs, o mesmo aparece com sintaxe especial. Auxiliando a implementação do arquivo makefile.

Capítulo 6

Bibliotecas

Apresenta-se neste capítulo um conjunto de programas auxiliares que são utilizados para montagens de bibliotecas no mundo GNU/Linux. A seguir apresenta-se um exemplo de montagem de biblioteca estática e um exemplo de montagem de biblioteca dinâmica.

6.1 Introdução a montagem de bibliotecas

Uma biblioteca é uma coleção de objetos (funções, classes, objetos) agrupados em um único arquivo. De um modo geral, um conjunto de arquivos com a extensão *.o, são reunidos para gerar um arquivo libNome.a (para biblioteca estática) ou libNome.so (para biblioteca dinâmica). O GNU/Linux tem um conjunto de programas auxiliares que são utilizados para criar, manter e gerenciar bibliotecas estáticas e dinâmicas. Apresenta-se a seguir uma breve descrição destes programas. Para obter informações detalhadas de cada programa dê uma olhada no *man page* ou nos manuais dos programas (os manuais podem ser baixados no site da gnu (<http://www.gnu.org>)).

6.1.1 ar

O programa ar é utilizado para manipular arquivos em um formato bem estruturado. O ar também cria tabelas com símbolos e referências cruzadas. O programa ar é que aglutina todos os objetos em uma lib, isto é, agrupa os arquivos *.o em uma lib. Veja a seguir o protótipo e um exemplo de uso do ar.

Protótipo e parâmetros do ar:

ar [opções] arquivos.

- t* *Lista os objetos da lib (biblioteca).*
- r* *Substitue funções quando necessário (arquivos antigos).*
- q* *Adiciona no modo append.*
- s* *Atualiza a tabela de símbolos.*
- c* *Cria o arquivo se este não existe.*
- v* *Modo verbose.*

Exemplo:

```
ar cru libNome.a arq1.o arq2.o arq3.o
```

Neste exemplo o programa ar vai juntar os arquivos arq1.o arq2.o arq3.o e gerar o arquivo libNome.a.

Veja a seguir a lista completa de opções do comando ar. A mesma pode ser obtida em seu sistema digitando **ar - -help**.

Listing 6.1: Saída do comando ar - -help.

```
Usage: ar [-X32_64] [-]{dmpqrstx}[abcfilNoPsSuvV] [member-name] [count] archive-
      file file...
      ar -M [<mri-script>]
commands:
d          - delete file(s) from the archive
m[ab]     - move file(s) in the archive
p         - print file(s) found in the archive
q[f]      - quick append file(s) to the archive
r[ab][f][u] - replace existing or insert new file(s) into the archive
t         - display contents of archive
x[o]      - extract file(s) from the archive
command specific modifiers:
[a]       - put file(s) after [member-name]
[b]       - put file(s) before [member-name] (same as [i])
[N]       - use instance [count] of name
[f]       - truncate inserted file names
[P]       - use full path names when matching
[o]       - preserve original dates
[u]       - only replace files that are newer than current archive contents
generic modifiers:
[c]       - do not warn if the library had to be created
[s]       - create an archive index (cf. ranlib)
[S]       - do not build a symbol table
[v]       - be verbose
[V]       - display the version number
[-X32_64] - (ignored)
Report bugs to bug-binutils@gnu.org and hjl@lucon.org
```

6.1.2 ranlib

Gera os índices para a biblioteca, isto é, gera um mapa de símbolos que serão utilizados pelos programas para localizar corretamente as funções a serem executadas.

Protótipo e parâmetros do ranlib:

ranlib [-v -V] arquivo.

-v *Versão do ranlib.*

Exemplo:

```
ranlib libNome.a
```

6.1.3 nm

Mostra os símbolos da biblioteca.

Protótipo e parâmetros do nm:

nm [opções] arquivo.

-C |--demangle Mostra nomes de forma clara para o usuário.

-S |--print-arnmap Imprime índice dos símbolos.

Exemplo:

```
nm libNome.a
```

Use o comando **nm -help** para obter as opções do comando nm.

Listing 6.2: Saída do comando nm -help .

Usage: nm [OPTION]... [FILE]...

List symbols from FILEs (a.out by default).

```
-a, --debug-syms      Display debugger-only symbols
-A, --print-file-name Print name of the input file before every symbol
-B                    Same as --format=bsd
-C, --demangle[={auto,gnu,lucid,arm,hp,edg,gnu-v3,java,gnat,compaq}]
                      Decode low-level symbol names into user-level names
    --no-demangle      Do not demangle low-level symbol names
    --demangler=<dso:function>
                      Set dso and demangler function
-D, --dynamic          Display dynamic symbols instead of normal symbols
    --defined-only     Display only defined symbols
-e                    (ignored)
-f, --format=FORMAT   Use the output format FORMAT.  FORMAT can be 'bsd',
                      'sysv' or 'posix'.  The default is 'bsd'
-g, --extern-only     Display only external symbols
-h, --help             Display this information
-l, --line-numbers     Use debugging information to find a filename and
                      line number for each symbol
-n, --numeric-sort     Sort symbols numerically by address
-o                    Same as -A
-p, --no-sort         Do not sort the symbols
-P, --portability      Same as --format=posix
-r, --reverse-sort     Reverse the sense of the sort
-s, --print-arnmap     Include index for symbols from archive members
    --size-sort        Sort symbols by size
-t, --radix=RADIX      Use RADIX for printing symbol values
    --target=BFDNAME   Specify the target object format as BFDNAME
-u, --undefined-only   Display only undefined symbols
-V, --version          Display this program's version number
-X 32_64              (ignored)
```

```
nm: supported targets: elf32-i386 a.out-i386-linux efi-app-ia32 elf32-little
elf32-big srec symbolsrec tekhex binary ihex trad-core
Report bugs to bug-binutils@gnu.org and hjl@lucon.org.
```

6.1.4 objdump

Imprime informações sobre as bibliotecas e objetos.

Protótipo e parâmetros do objdump:

objdump [opções][parâmetros]

-d, -debugging.
-syms Tabela de símbolos.
-a Informações arquivo.

Exemplo:

```
objdump -file-header file.o
```

Dica: Para maiores detalhes, execute **objdump - -help**.

6.1.5 ldd

Lista as bibliotecas dinâmicas que determinado programa usa.

Protótipo e parâmetros do ldd:

ldd [-d-r] programa

- -help Imprime um help.
- -version Imprime a versão do ldd.
-d,- -data-relocs Processa uma realocação dos dados.
r,- -function-relocs Processa uma realocação dos dados e funções.
-v,- -verbose Imprime informações em geral.

Exemplo:

```
ldd /bin/netcape
```

Veja a seguir a saída do comando `ldd /usr/bin/lyx`. Lembre-se que LyX é o editor utilizado para montar esta apostila. Observe o uso das bibliotecas libXForms, Xpm, X11 e libstdc++.

Listing 6.3: Saída do comando `ldd /usr/bin/lyx`.

```
libforms.so.0.88 => /usr/lib/libforms.so.0.88 (0x40032000)
libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4 (0x400b4000)
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x400c3000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x400cc000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x400e3000)
libstdc++-libc6.2-2.so.3 => /usr/lib/libstdc++-libc6.2-2.so.3 (0
x401d9000)
libm.so.6 => /lib/i686/libm.so.6 (0x4021c000)
libc.so.6 => /lib/i686/libc.so.6 (0x4023f000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

6.1.6 ldconfig

O programa ldconfig determina os links necessários em tempo de execução para bibliotecas compartilhadas (shared libs)¹.

Protótipo e parâmetros do ldconfig:

`ldconfig [-p-v]libs`

- p Mostra bibliotecas compartilhadas.
- v Modo verbose.

6.2 Convenção de nomes para bibliotecas

- O nome de uma biblioteca deve iniciar com lib.
- A extensão será *.a para bibliotecas estáticas.
- A extensão será *.so para bibliotecas dinâmicas.
- Versões:
libNome.so.VersãoMaior.VersãoMenor.Patch
A versão maior é uma versão incompatível com as demais.
A versão menor inclui novidades.
A patch inclui correções de bugs.
- Uma biblioteca que tem o nome encerrado com -g contém instruções de debugagem.
- Uma biblioteca que tem o nome encerrado com -p contém instruções para o profiler (gprof).

6.3 Bibliotecas usuais

Apresenta-se na Tabela ?? algumas bibliotecas usuais.

6.4 Montando uma biblioteca estática (libNome.a)

Para utilizar a biblioteca o programador precisa dos arquivos com o cabeçalho (formato como os objetos e as funções foram construídos) e o arquivo da lib.

Seqüência para criar uma biblioteca estática no GNU/Linux:

1. Cria o arquivo de cabeçalho *.h (declaração das funções em C e das classes em C++)

```
emacs Tponto.h
Tponto.h class TPonto { .....};
//ctrl+x ctrl+s para salvar
//ctrl+x ctrl+c para sair
```

¹Descrito no livro Linux Unleashed, não disponível em minha máquina.

Tabela 6.1: Bibliotecas usuais.

Biblioteca	Include	Uso
libGL.so	<GL/gl.h>	Interface para OpenGL
libjpeg.so	<jpeglib.h>	Interface para arquivos jpeg
libpbm.so	<pbm.h>	Interface para bitmaps monocromáticos
libpgm.so	<pgm.h>	Interface para bitmaps tons de cinza
libpng.so	<png.h>	Interface para arquivos portable bitmap format
libpnm.so	<pnm.h>	Interface para bitmaps pbm, ppm, pgm
libpthread.so	<pthread.h>	Posix Threads
libvga.so	<vga.h>	Acesso a tela vga
libz.so	<zlib.h>	Biblioteca para compactação de arquivos
glibc		Biblioteca padrão C
magick++	<magick++.h>	Biblioteca gráfica

2. Cria o arquivo de código *.cpp (definição das funções)

```
emacs TPonto.cpp
/*Define funções da classe*/ ....
```

3. Compila os arquivos de código (*.cpp) gerando os arquivos (*.o)

```
g++ -c TPonto.cpp
```

4. Cria a biblioteca (a lib)

```
ar -q libNome.a TPonto.o
```

5. Publica a biblioteca com

```
ranlib libTPonto.a
```

Observe que os itens 1 e 2 se referem a edição dos arquivos do programa. Em 3, o programa é compilado. Pode-se utilizar um arquivo makefile para automatizar esta tarefa. Em 4, cria-se a biblioteca, gerando-se o arquivo libTPonto.a. Em 5 a biblioteca é publicada no sistema.

6.4.1 Usando uma biblioteca estática

No exemplo a seguir compila o programa Prog.cpp e pede para linkar em modo estático (-static) a biblioteca nomeLib que esta localizada em pathLib.

```
Exemplo:
g++ Prog.cpp -static -LpathLib -lnomeLib
```

6.5 Montando uma biblioteca dinâmica (libNome.so)

Roteiro para criar uma biblioteca dinâmica no GNU/Linux:

1. Cria o arquivo de cabeçalho *.h (declaração das funções e classes)

```
emacs TPonto.h
class TPonto { .....};
```

2. Cria o arquivo de código *.cpp (definição das funções)

```
emacs TPonto.cpp
/*Define funcoes da classe*/
```

3. Compila os arquivos de código (*.cpp) gerando os arquivos *.o
A opção -fPIC, gera o código com posicionamento independente, podendo o mesmo ser carregado em qualquer endereço.

```
g++ -fPIC -c TPonto.cpp -o TPonto.o
```

4. Cria a biblioteca dinâmica
A opção -WL passa informações para o linker ld.

```
g++ -shared -Wl,-soname,TPonto.so.1 -o libTPonto.so.1.0 TPonto.o
```

5. Copia a lib para /usr/local/lib (como root)

```
cp libTPonto.so.1.0 /usr/local/lib
```

6. Pode-se criar links simbólicos para a lib

```
cd /usr/local/lib/
ln -s libTPonto.so.1.0 libTPonto.so.1
ln -s libTPonto.so.1.0 libTPonto.so
```

7. Publica a lib
(#include na tabela de bibliotecas dinâmicas, cria link, e include em /etc/ld.so.cache)

```
/sbin/ldconfig
```

6.5.1 Usando uma biblioteca dinâmica

O exemplo ilustrado a seguir inclui o arquivo `#include <dlfcn.h>`, um arquivo de inclusão de bibliotecas dinâmicas. Veja no exemplo outras informações (extraído da fonte [5]).

```
#include <dlfcn.h>
#include <fstream.h>....
main()
{
    //Cria ponteiro para a lib
    void* ptrLib;
    //Cria ponteiro para função da lib
    void (*ptrFuncaoLib)();
    //Carrega a lib
    //dlopen(const char* fileName, int flag);
    ptrLib = dlopen("nomeLib.so.1.0", RTLD_LAZY);
    //Verifica se não tem erro com a função dlerror
    //const char* dlerror();
    cout << dlerror();
    //Obtém endereço da função
    //void* dlsym(void* handle, char* simbolo);
    ptrFuncaoLib = dlsym(ptrLib, "NomeFuncaoNaLib");
    //Usa a função
    int x = (*ptrFuncaoLib)();
    //Fecha a lib
    //int dlclose(void * handle);
    dlclose(ptrLib);
}
```

6.5.2 Vantagens/desvantagens da biblioteca dinâmica

- Em uma biblioteca estática o programa é maior porque inclui todas as bibliotecas.
- Quando o programa é linkado com bibliotecas dinâmicas, o mesmo fica menor, pois as bibliotecas são carregadas em tempo de execução.
- Um programa que faça uso de uma biblioteca dinâmica não precisa ser recompilado se ocorrerem modificações na biblioteca. Só será necessário modificar o programa que acessa a biblioteca dinâmica se o mesmo utilizar objetos da biblioteca que sofreram modificações na sua interface.

Veja a seguir um arquivo makefile para gerar o programa `e87-Polimorfismo.cpp`, anteriormente apresentado. Parte da listagem inclui instruções para o programa `libtool` que será apresentado no Capítulo ??.

Listing 6.4: Arquivo makefile com bibliotecas estáticas e dinâmicas.

```
#=====DEFINICOES=====
ARQUIVOS= e87-TCirculo.cpp e87-TElipse.cpp e87-TPonto.cpp e87-Polimorfismo.
cpp
```

```

OBJETOS= e87-TCirculo.o e87-TElipse.o e87-TPonto.o
DIRINCLUDE = -I/usr/include/g++ -I/usr/include
DIRLIBD = /home/andre/Andre/ApostilasPessoais/ApostilaProgramacao/Exemplos/
        cursocpp/Cap-GNU/biblioteca
PARAMETROSLIB= -lm
COMPILADOR = g++
LIBS = TPonto
LIBD = TPonto
PROG = e87-Polimorfismo.cpp

#-----Lista de opções.
#list:
#      "obj:      Gera objetos comuns
#      exe:      Gera executável comum"
#      all:      obj exe
#      libs:     Gera biblioteca estática
#      exes:     Gera executável usando biblioteca estática
#      alls:     libs exelibs
#      libd:     Gera biblioteca dinâmica
#      exed:     Gera executável usando biblioteca dinâmica
#      libs_libtool: Gera biblioteca estatica usando lib_tool
#      exes_libtool: Gera executável usando biblioteca estatica e libtool
#      init_doc:  Inicializa o doxygen
#      doc:      Gera documentação a partir de código documentado
#      clean:    Apaga arquivos "

#-----Compilação padrão
all : obj exe

obj : $(ARQUIVOS)
      $(COMPILADOR) -c $(ARQUIVOS) $(DIRINCLUDE) $(DIRECL)

exe : $(PROG) $(OBJETOS)
      $(COMPILADOR) $(PROG) $(OBJETOS) $(DIRINCLUDE) $(DIRETORIOLIB) $(
        PARAMETROSLIB) -o e87-Polimorfismo

#-----Criando biblioteca estática
alls : libs exelibs

libs : $(OBJETOS)
      ar -q libTPonto.a $(OBJETOS)
#ar -cru libTPonto.a $(OBJETOS)
      ranlib libTPonto.a #publica a lib

#Criando executavel usando a biblioteca estática
exelibs : libs
      $(COMPILADOR) e87-PolimorfismoStatic.cpp -static -L. -l$(LIBS) -o e87-
        PolimorfismoStatic

#-----Criando biblioteca dinâmica
alld : objd libd

objd : $(ARQUIVOS)
      $(COMPILADOR) -fPIC -c $(ARQUIVOS) $(DIRINCLUDE) $(DIRECL)

```

```

libd : $(OBJETOS)
      $(COMPILADOR) -shared -Wl,-soname,TPonto.so.1 -o libTPonto.so.1.0 $(
      OBJETOS)
      ln -s libTPonto.so.1.0 libTPonto.so.1
      ln -s libTPonto.so.1.0 libTPonto.so
      /sbin/ldconfig          #publica a biblioteca (como root)

#-----Criando executavel usando a biblioteca dinâmica
exelibd : e87-PolimorfismoDinamic.cpp libd
          $(COMPILADOR) e87-PolimorfismoDinamic.cpp -L$(DIRLIBD) -l$(LIBD) -o e87-
          PolimorfismoDinamic

#-----Limpeza.
clean :
        rm -f *.o *.obj *.so* a.out e87-PolimorfismoDinamic e87-
        PolimorfismoStatic *.*~ ~~ libTPonto.*

#=====USANDO LIBTOOL=====
#Para ficar portátil, todas as etapas devem ser realizadas com o libtool
#obj_libtool : $(ARQUIVOS)
#          libtool $(COMPILADOR) -c $(ARQUIVOS)

#-----Criando biblioteca estatica usando o libtool
libs_libtool: $(ARQUIVOS)
              libtool $(COMPILADOR) -o libTPonto.a $(ARQUIVOS)

#-----Criando executavel usando libtool com biblioteca estatica
#Incompleto
exes_libtool: e87-PolimorfismoStatic.cpp libs_libtool
              libtool $(COMPILADOR) e87-PolimorfismoStatic.cpp -static -L. -l$(LIBS) -
              o e87-PolimorfismoStatic-libtool

#-----Criando biblioteca dinâmica usando o libtool
libd_libtool: $(ARQUIVOS)
              libtool $(COMPILADOR) -o libTPonto.la $(ARQUIVOS)

#-----Criando executavel usando libtool com biblioteca dinâmica
#Incompleto
exed_libtool: e87-PolimorfismoStatic.cpp libd_libtool
              libtool $(COMPILADOR) e87-PolimorfismoDinamic.cpp libTPonto.la -o e87-
              PolimorfismoDinamic-libtool

#-----Criando documentação com doxygen
init_doc:
          doxygen -g doxygen.config

doc : $(ARQUIVOS)
      doxygen doxygen.config

```

Faça cópias do e87-Polimorfismo.cpp criando os arquivos e87-PolimorfismoStatic.cpp e e87-PolimorfismoDinamic.cpp e então execute a sequência ilustrada a seguir.

Listing 6.5: Arquivo mostrando o uso do makefile.

```
[root@mercurio biblioteca]# make clean
rm -f *.o *.obj *.so* a.out e87-PolimorfismoDinamic e87-PolimorfismoStatic
*.~*~ libTPonto.*

[root@mercurio biblioteca]# make all
g++ -c e87-TCirculo.cpp e87-TElipse.cpp e87-TPonto.cpp e87-Polimorfismo.cpp -
-I/usr/include/g++ -I/usr/include
g++ e87-Polimorfismo.cpp e87-TCirculo.o e87-TElipse.o e87-TPonto.o -I/usr/
include/g++ -I/usr/include -lm -o e87-Polimorfismo

[root@mercurio biblioteca]# make alls
ar -q libTPonto.a e87-TCirculo.o e87-TElipse.o e87-TPonto.o
ranlib libTPonto.a #publica a lib
g++ e87-PolimorfismoStatic.cpp -static -L. -lTPonto -o e87-PolimorfismoStatic

[root@mercurio biblioteca]# make alld
g++ -fPIC -c e87-TCirculo.cpp e87-TElipse.cpp e87-TPonto.cpp e87-Polimorfismo.
cpp -I/usr/include/g++ -I/usr/include
g++ -shared -Wl,-soname,TPonto.so.1 -o libTPonto.so.1.0 e87-TCirculo.o e87-
TElipse.o e87-TPonto.o
ln -s libTPonto.so.1.0 libTPonto.so.1
ln -s libTPonto.so.1.0 libTPonto.so
/sbin/ldconfig #publica a biblioteca (como root)

[root@mercurio biblioteca]# make libs_libtool
libtool g++ -o libTPonto.a e87-TCirculo.cpp e87-TElipse.cpp e87-TPonto.cpp e87
-Polimorfismo.cpp
ar cru libTPonto.a
ranlib libTPonto.a

[root@mercurio biblioteca]# make libd_libtool
libtool g++ -o libTPonto.la e87-TCirculo.cpp e87-TElipse.cpp e87-TPonto.cpp
e87-Polimorfismo.cpp
rm -fr .libs/libTPonto.la .libs/libTPonto.* .libs/libTPonto.*
ar cru .libs/libTPonto.al
ranlib .libs/libTPonto.al
creating libTPonto.la
(cd .libs && rm -f libTPonto.la && ln -s ../libTPonto.la libTPonto.la)
```

6.6 Sentenças para bibliotecas

- O padrão para o nome da lib é: libNome.so.versao.subversao.release.
- Você pode acrescentar novas paths para bibliotecas dinâmicas modificando a variável de ambiente LD_LIBRARY_PATH.

Capítulo 7

Libtool

Neste capítulo apresenta-se o libtool, mais um pacote da gnu, utilizado para facilitar o desenvolvimento de programas em múltiplas plataformas.

7.1 Introdução ao libtool²

Como será descrito nos capítulos seguintes, o libtool é mais um programa da gnu que facilita o desenvolvimento de bibliotecas multiplataforma. O mesmo é usado para desenvolvimento de bibliotecas no ambiente GNU/Linux.

Vantagens do uso do libtool

- Maior elegância.
- Integrado ao autoconf e automake.
- Maior portabilidade.
- Trabalha com bibliotecas estáticas e dinâmicas.

7.2 Forma de uso do libtool

Como faço para usar o libtool ?

De um modo geral, basta digitar o comando **libtool** seguido do comando que você usaria para compilar seu programa ou biblioteca.

Exemplos de uso do libtool estão listados no diretório do libtool.

Veja a seguir a saída do comando **libtool - - help**.

Listing 7.1: Arquivo libtool - -help.

```
Usage: libtool [OPTION]... [MODE-ARG]...
```

```
Provide generalized library-building support services.
```

```
--config          show all configuration variables
--debug           enable verbose shell tracing
-n, --dry-run     display commands without modifying any files
```

<code>--features</code>	display basic configuration information and exit
<code>--finish</code>	same as <code>'--mode=finish'</code>
<code>--help</code>	display this help message and exit
<code>--mode=MODE</code>	use operation mode MODE [default=inferred from MODE-ARGS]
<code>--quiet</code>	same as <code>'--silent'</code>
<code>--silent</code>	don't print informational messages
<code>--version</code>	print version information

MODE must be one of the following:

<code>clean</code>	remove files from the build directory
<code>compile</code>	compile a source file into a libtool object
<code>execute</code>	automatically set library path, then run a program
<code>finish</code>	complete the installation of libtool libraries
<code>install</code>	install libraries or executables
<code>link</code>	create a library or an executable
<code>uninstall</code>	remove libraries from an installed directory

MODE-ARGS vary depending on the MODE. Try `'libtool --help --mode=MODE'` for a more detailed description of MODE.

7.3 Criando uma biblioteca sem o libtool

Reveja a seguir como criar uma biblioteca estática sem uso do libtool.

Exemplo:

```
ar cru libNome.a a.o b.o c.o
ranlib libNome.a
```

Para criar uma biblioteca estática usando o arquivo makefile anteriormente apresentado execute o comando:

Exemplo:

```
make clean
make libs
```

A saída gerada pelo makefile é dada por:

```
[andre@mercurio libtool-biblioteca]$ make libs
g++ -c -o e87-TCirculo.o e87-TCirculo.cpp
g++ -c -o e87-TElipse.o e87-TElipse.cpp
g++ -c -o e87-TPonto.o e87-TPonto.cpp
ar -q libTPonto.a e87-TCirculo.o e87-TElipse.o e87-TPonto.o
ranlib libTPonto.a
```

7.4 Criando uma biblioteca estática com o libtool

Veja a seguir como usar o libtool para gerar a biblioteca estática.

Exemplo:

```
libtool g++ -o libTPonto.a e87-TPonto.cpp
                        e87-TElipse.cpp e87-TCirculo.cpp
```

Veja a seguir a saída gerada pelo libtool.

```
mkdir .libs
ar cru libTPonto.a
ranlib libTPonto.a
```

7.5 Criando uma biblioteca dinâmica com o libtool

O mesmo exemplo usando o libtool e biblioteca dinâmica. Observe que a única alteração é o nome da biblioteca, que agora se chama libTPonto.la.

Exemplo:

```
libtool g++ -o libTPonto.la e87-TPonto.cpp
                        e87-TElipse.cpp e87-TCirculo.cpp
```

Veja a seguir a saída gerada pelo libtool.

```
rm -fr .libs/libTPonto.la .libs/libTPonto.* .libs/libTPonto.*
ar cru .libs/libTPonto.al
ranlib .libs/libTPonto.al
creating libTPonto.la (cd .libs && rm -f libTPonto.la
                        && ln -s ../libTPonto.la libTPonto.la)
```

7.6 Linkando executáveis

Formato usual:

Exemplo:

```
g++ -o nomeExecutável nomePrograma.cpp libNome.la
```

Formato usando o libtool:

Exemplo:

```
libtool g++ -o nomeExecutável nomePrograma.cpp libNome.a
```

7.7 Instalando a biblioteca

Formato usual:

Exemplo:

```
//como root
cp libNome.a /usr/lib
ranlib /usr/lib/libNome.a
```

Formato usando o libtool:

```
Exemplo:
libtool cp libNome.a /usr/lib/
//ou
libtool install -c libNome.a /usr/lib/libNome.la
//ou
libtool install -c .libs/libNome.a /usr/lib/libNome.so.0.0
libtool install -c .libs/libNome.a /usr/lib/libNome.la
libtool install -c .libs/libNome.a /usr/lib/libNome.a
```

7.8 Modos do libtool

Para saber mais sobre o funcionamento de cada um dos módulos abaixo listados, execute o comando: **libtool - -help - -mode = MODO**.

Compilação, o libtool atua chamando o compilador do sistema:

libtool - -help - -mode=compile

Linkagem, o libtool atua executando a linkagem:

libtool - -help - -mode=link

Instalação, o libtool atua instalando o programa:

libtool - -help - -mode=install

Execução, o libtool atua executando o programa:

libtool - -help - -mode=execute

Desinstalação, o libtool atua desinstalando o programa:

libtool - -help - -mode=uninstall

7.9 Sentenças para o libtool

- Para executar o gdb com o libtool use:

```
libtool gdb nomePrograma
```

- Para obter informações do libtool:

```
libtool - -help
```

- Para gerar apenas bibliotecas estáticas, passar o flag:

```
--disable-shared
```

- Durante o desenvolvimento costuma-se usar biblioteca estática com opção de debugagem.

- Em 9/2001 o libtool ainda não era totalmente compatível com C++.
- Leia o livro “GNU AUTOCONF, AUTOMAKE, AND LIBTOOL” disponível gratuitamente no site (<http://sources.redhat.com/autobook/>).
- Bibliotecas compartilhadas usam a especificação PIC (position independent code).

Capítulo 8

Debug (Depuradores, Debuggers)

Apresenta-se neste capítulo informações sobre o debugger da gnu, o gdb.

8.1 Introdução ao debugger

Bem, você é um bom programador, mas..., ainda existem alguns probleminhas, e você terá de rastrear o seu código para eliminar aqueles pequenos insetos. Não adianta, você vai ter de debugar seu código.

Mas afinal de contas o que é e para que serve um debugger ?

Um debugger é um programa que monitora a execução de outros programas. Com o debugger você pode acompanhar a execução do programa e identificar onde o mesmo esta com problemas. Para tal, o debugger possibilita a avaliação do conteúdo das variáveis e o acompanhamento passo a passo da execução do programa.

O GNU/Linux, Unix tem o gdb, um debug em modo texto e seus frontends o xgdb e o kdbg.

Antes de mais nada, para poder debugar o seu código, você precisa acrescentar as informações para o debug passando a opção de compilação -g (CPPFLAGS= -g). Desta forma o gdb poderá examinar o seu executável (ou o arquivo core) para verificar o que aconteceu.

8.2 Comandos do gdb

Apresenta-se na Tabela ?? uma lista com os comandos do gdb.

8.3 Exemplo de uso do gdb

Um pequeno exemplo de uso do gdb.

Exemplo

```
(gdb) Run           //Roda o programa
(gdb) backtrace      //Mostra a pilha (o último comando executado)
(gdb) break 23       //Acrescenta breakpoint na linha 23
(gdb) list           //Mostra o código fonte perto do breakpoint
(gdb) p var          //Mostra o conteúdo da variável
(gdb) c              //Continua execução
```

Tabela 8.1: Comandos do gdb.

Comando	Ação
<code>gdb</code>	Executa o debugger.
<code>run prog</code>	Executa o programa prog.
<code>run prog arg</code>	Roda o programa com os argumentos.
<code>bt</code>	Apresenta um rastreamento da pilha.
<code>break func</code>	Cria breakpoint na função func.
<code>list arq.cpp</code>	Visualiza o arq.cpp.
<code>break 25 (ou b25)</code>	Acrescenta breakpoint na linha 25.
<code>delete (d)</code>	Deleta os breakpoint.
<code>c</code>	Continua.
<code>step</code>	Executa um passo.
<code>step 10</code>	Executa os próximos 10 passos.
<code>next</code>	Executa uma linha.
<code>next 12</code>	Executa as próximas 12 linhas.
<code>print var</code>	Imprime o conteúdo da variável.
<code>what atributo</code>	Mostra conteúdo do atributo/variável.
<code>quit</code>	Abandona o debug.
<code>help com</code>	Help sobre o comando.

8.4 Sentenças para o gdb

- No GNU/Linux, quando um programa trava ou é encerrado de forma inesperada, é gerado um arquivo core. O arquivo core pode ser aberto pelo gdb para localizar a posição onde o programa travou.
- No gnu você pode simplificar o trabalho de debugagem incluindo a macro `__FUNCTION__` que imprime o nome da função que esta sendo executada.

Exemplo:

```
cout << " na função : " << __FUNCTION__ << endl;
```

Capítulo 9

Profiler (gprof)²

Neste capítulo apresenta-se brevemente o gprof, o profiler da gnu.

9.1 Introdução ao gprof

Um profiler é um programa utilizado para avaliar o desempenho do seu programa, permitindo encontrar os gargalos (pontos onde o programa demora mais).

O profiler apresenta um gráfico com o tempo de execução de cada função.

Primeiro compila o programa incluindo a opção -pg. Criando o executável a.out que é aberto pelo gprof

```
g++ -pg ex-funcaoobjeto1.cpp
```

A seguir executa o gprof

```
gprof --brief -p
```

Veja abaixo a saída gerada pelo gprof

```
[andre@mercurio Cap4-STL]$ gprof --brief -p
Flat profile:
Each sample counts as 0.01 seconds. no time accumulated
% cumulative self self total time seconds seconds calls Ts/call Ts/call name
0.00 0.00 0.00 28 0.00 0.00 _Deque_iterator<int, int &, int *, 0>::~_S_buffer_size(void)
0.00 0.00 0.00 21 0.00 0.00 _Deque_iterator<int, int &, int *, 0>::~operator!=
(_Deque_iterator<int, int &, int *, 0> const &) const
0.00 0.00 0.00 17 0.00 0.00 _Deque_iterator<int, int &, int *, 0>::~operator-
(_Deque_iterator<int, int &, int *, 0> const &) const
0.00 0.00 0.00 15 0.00 0.00 _Deque_base<int, allocator<int>, 0>::~~Deque_base(void)
0.00 0.00 0.00 14 0.00 0.00 void destroy<int *>(int *, int *) 0.
```

Observe a direita o nome da função e a esquerda o tempo de execução.

9.2 Sentenças para o profiler:

- Para aprender a usar o gprof, baixe e leia o manual do gprof no site da gnu.

- Se você quer um compilador mais rápido e usa máquinas com processadores pentium pense em usar o pgcc. Um compilador descendente do compilador da gnu e otimizado para processadores pentium. Procure por pgcc na internet.
- Você só deve se preocupar com performance (e com o gprof) depois que for um bom programador. Primeira faça com que os programas funcionem, a seguir se preocupe com bugs, faça a documentação,....., depois de tudo se preocupe com a performance.

Capítulo 10

Versão de Depuração, Final e de Distribuição

Neste capítulo apresenta-se as opções para criar a versão de depuração e a versão final de seu programa. Apresenta-se ainda as formas de distribuição de programas.

10.1 Versão debug, release e de distribuição

A medida que o programa é desenvolvido e os bugs corrigidos, costuma-se trabalhar com uma versão de debugagem (passando a opção -g para o compilador). Depois, quando deseja-se distribuir um release do programa, eliminam-se todas as opções de debugagem e colocam-se opções de otimização.

10.1.1 Versão debug

1. Ativar a opção de debugagem (-g).
2. Ativar todos os warnings (-Wall).

10.1.2 Versão final (release)

1. Desativar todas as opções de debugagem (tirar -g).
2. Ativar as opções de otimização (-O1,-O2,-O3).
3. Ativar todos os warnings (-Wall).

10.1.3 Distribuição dos programas e bibliotecas

Uma biblioteca pode ser vendida, distribuindo-se os arquivos de cabeçalho (*.h) e os arquivos da biblioteca (*.lib).

Um programador que comprou as bibliotecas, pode usar as funções e objetos da biblioteca consultando os manuais e os arquivos de cabeçalho (*.h).

Observe que como os arquivos *.cpp não são distribuídos, o programador não tem acesso a forma como as funções foram implementadas, isto é, não tem como avaliar a qualidade da biblioteca. Este é o formato de distribuição de bibliotecas no ambiente Windows. Um sistema proprietário que esconde de quem compra o programa o seu código, o seu real funcionamento (suas qualidades e seus problemas).

Um formato mais moderno e democrático é distribuir tanto os arquivos de cabeçalho (*.h) como os de implementação (*.cpp), este é o sistema de distribuição do GNU/Linux.

Para distribuir seu código você pode utilizar uma das opções descritas a seguir.

Distribuir o seu código fonte em um arquivo .tar.gz

1. Gera o arquivo de distribuição

```
tar -cvzf nomeArquivo.tar.gz path_do_programa
```

2. Permite o acesso dos usuários pela internet ou pela distribuição de disketes (zip, cd).

Distribuir o seu código fonte com patches (atualizações)

Além de distribuir o seu código com o arquivo .tar.gz você pode distribuir upgrades, isto é, distribuições que acrescentam apenas as modificações que foram realizadas no código. A grande vantagem é que o usuário precisa baixar arquivos pequenos. O roteiro abaixo mostra como distribuir atualizações de código com patches.

Programador_etapa_1:

1. Gera o programa

```
make  
testa se o programa esta ok...
```

2. Gera a distribuição .tar.gz

```
//se o arquivo makefile foi configurado para gerar uma distribuição  
make dist  
//ou cria um arquivo compactando todo o código com o tar  
tar -cvzf prog.tar.gz path_do_programa
```

Usuário_etapa_1:

1. Baixa, descompacta, compila e instala o programa.

```
ftp site_com_o_programa  
login  
get prog.tar.gz  
quit  
tar -xvzf prog.tar.gz  
cd prog  
./configure  
make  
make install
```

Programador_etapa_2:

1. Faz atualizações no código (upgrades)

```
Edita os arquivos com o programa e inclui atualizações  
e correções de bugs
```

2. Gera os arquivos com as diferenças (coisas novas nos códigos)

```
diff arq1-old.cpp arq1.cpp > arq1.dif
```

3. Gera o pacote com as atualizações

```
tar -cvzf path-versaoold-versaonova.tar.gz *.dif
```

Usuário_etapa_2:

1. Baixa e descompacta o arquivo path-versaoold-versaonova.tar.gz.

```
ftp site_com_o_programa  
login  
get path-versaoold-versaonova.tar.gz  
tar -xvzf path-versaoold-versaonova.tar.gz
```

2. Executa o programa patch para que o código antigo (arq1.cpp) seja alterado, incluindo as alterações da nova versão, ou seja, gera um novo arquivo arq1 somando ao arquivo arq1 as novidades listadas no arquivo arq1.dif.

```
cp arq1.cpp arq1.cpp~  
patch arq1.cpp arq1.dif
```

3. Configura, compila e instala o programa

```
./configure  
make  
make install
```

4. As alterações realizadas no arquivo arq1.cpp podem ser revertidas (voltar a ser o que era) executando-se

```
patch -r arq1.cpp arq1.dif
```

Distribuir o seu código fonte através do uso do CVS

Você terá de ter um servidor de CVS instalado e funcionando. Deve criar um repositório para o seu programa (com releases,...) e um sistema que permita aos usuários baixar os arquivos no servidor CVS para serem compilados.

Veja exemplo de como baixar um programa de terceiros usando o cvs na seção ??.

Distribuir o programa (e os fontes) em pacotes rpm

O desenvolvimento de arquivo rpm é um pouco complexo e esta fora do escopo desta apostila. Alguns ambientes de desenvolvimento, como o kdevelop, geram os arquivos rpm para você.

10.2 Sentenças para distribuição de código fonte

- Se você esta programando no GNU/Linux é bem provável que já tenha baixado e compilado o kernel do Linux. O mesmo é disponibilizado no site <http://kernel.org>. Vá ao site do kernel do Linux e compare o tamanho das atualizações do kernel, isto é, compare os arquivos kernel-versao.tar.bz2 e patch-versao.bz2. Veja o resultado aí em baixo, de 23576Kb para 807Kb.

patch-2.4.18.bz2 807 KB 25-02-2002 19:44:00

linux-2.4.18.tar.bz2 23596 KB 25-02-2002 19:40:00 File

- Use o patch, a atualização dos programas fica muito mais rápida pois os arquivos de atualizações são pequenos e podem ser baixados com extrema rapidez na internet.
- Já que estamos falando do uso de patch para baixar arquivos menores pela internet, pense em usar mais a internet como um instrumento de aprendizado, de trabalho e de troca de dados. Evite span, evite baixar e mandar via email fotos e filmes com pouca utilidade.

Capítulo 11

Documentação de Programas Usando Ferramentas GNU/Linux

Apresenta-se neste capítulo o uso do formato JAVA_DOC para embutir documentação em seus programas. O uso do programa doxygen para gerar a documentação em diferentes formatos (html, tex, rtf). Cita-se ainda os formatos sgml e xml para geração de manuais profissionais.

11.1 Introdução a documentação de programas

A documentação é uma etapa fundamental para compreensão, aperfeiçoamento e manutenção de programas.

Existem atualmente alguns programas e padrões para documentação de códigos em C++. Vou descrever brevemente o formato JAVA_DOC que é aceito pelo gerador de documentação DOXYGEN.

Como funciona ?

Você inclui em seu código um conjunto de tags. Estes tags não interferem na compilação de seu programa, apenas incluem informações que serão identificadas por um programa externo (o doxygen), para gerar a documentação das classes, dos atributos e dos métodos.

- A primeira etapa é incluir a documentação nos arquivos *.h e *.cpp, veja seção ??.
- A segunda etapa consiste em executar o doxygen (que já foi instalado e configurado) para gerar a documentação, veja seção ??.

11.2 Documentação embutida no código com JAVA_DOC

O formato JAVA_DOC é amplamente utilizado para documentação de programas em C++, também é válido para documentação de códigos em JAVA e IDL.

11.2.1 Exemplo de código documentado

Para inserir um breve comentário utilize três barras invertidas.

```
///  
///Breve comentário (apenas uma linha).
```

Para inserir um breve comentário e um comentário mais detalhado use

```
/** Breve comentário
 * Comentário mais detalhado
 * .....
 * @class TNome
 * @file NomeArquivo
 */
class TNome
{
.
```

No exemplo acima observe a posição do breve comentário e do comentário detalhado. Veja ainda a forma de definição do nome da classe e do arquivo.

Para inserir um comentário embutido use:

```
int a; /**< Comentário pequeno, embutido*/
```

Observe que o comentário inicia com um `/**<` e termina com um `*/`. Veja a seguir outro exemplo.

```
/** Um enumerador (breve descrição)
 * Descrição detalhada do enumerador
 */
enum ENome {
segunda, /**< Comentário pequeno, embutido*/
terca,   /**< Comentário pequeno, embutido*/
}
/** Breve descrição da função
 * Descrição detalhada da função
 * @param int a
 * @param int b
 * @return retorna a soma (int)
 * @see
 */
int Soma(int a, int b)
{return a+b;};
```

No exemplo acima informa-se o nome dos parâmetros e de retorno da função soma.

11.2.2 Sentenças para documentação `java_doc`

- Se houver documentação duplicada: na classe `/*.h` e nos arquivos de definição `/*.cpp`. Vai usar a breve descrição incluída na classe `/*.h` e a documentação detalhada incluída na definição da função `/*.cpp`.
- Aceita os tags:
class, struct, union, enum, fn(função), var (atributos), def (define), file(arquivo), namespace (um namespace), brief (breve comentário).
- Para inserir uma lista:

```
/**
```

```

*           Nome Lista
           -Ítem      A
                   -#      SubÍtem A.1
                   -#      SubÍtem A.2
           -Ítem      B
                   -#      SubÍtem B.1
                   -#      SubÍtem B.2

*/

```

- Para criar grupos:
`/** @addgroup <NomeGrupo> */`
`/** @ingroup NomeGrupo */`
`/** @defgroup NomeGrupo */`
- Para detalhes, veja a documentação do formato JAVA_DOC.

11.3 Tutorial de configuração e uso do DOXYGEN

O doxygen é um programa que gera a documentação (API) a partir de informações incluídas no código. As informações são incluídas no código utilizando-se o formato JAVA_DOC (veja seção ??).

Breve tutorial de uso do doxygen

1. Baixar o programa doxygen (ou usar os CDs da sua distribuição GNU/Linux).
Atualmente a maioria das distribuições GNU/Linux inclui o doxygen.
Você pode obter o doxygen no site <http://www.stack.nl/~dimitri/doxygen/>.
2. Instalar o doxygen usando o arquivo .tar.gz (`./configure && make && make install`), ou usando pacote rpm.
3. Criar um arquivo de configuração do projeto a ser documentado.
`doxygen -g nomeArquivoConfiguracao`
4. Incluir a documentação em seu código (veja seção ??).
5. Executar o doxygen (gera por default saída html).
`doxygen nomeArquivoConfiguracao`
6. Para gerar saída latex setar a variável
`GENERATE_LATEX = YES`
Vá para o diretório com os arquivos do latex e execute
`(make && make ps && make pdf)`.
Para gerar 2 folhas por página, vá para o diretório
com os arquivos do latex e execute:
`(make && make ps_2on1 && make pdf_2on1)`.
7. Você pode adicionar ao doxygen o programa da graphviz. O mesmo é usado para gerar diagramas de relacionamento das diversas classes. Procure pelo programa no site (<http://www.research.att.com/sw/tools/graphviz/>).

Apresenta-se nas linhas abaixo um arquivo de configuração do doxygen. O arquivo inclui comentários dentro de parenteses.

Listing 11.1: Exemplo de arquivo de configuração do doxygen com comentários.

```
..# Doxyfile 0.1
#-----
# General configuration options
#-----
PROJECT_NAME           = "Biblioteca de objetos - LIB_LMPT" (nome projeto)
PROJECT_NUMBER         = 0.4 (versão)
OUTPUT_DIRECTORY       = /home/andre/Andre/Desenvolvimento/LIB_LMPT-api/ (diretório de saída)
OUTPUT_LANGUAGE        = English (linguagem)
EXTRACT_ALL            = YES (extrair todas as informações)
EXTRACT_PRIVATE        = YES (incluir atributos/funções privados)
EXTRACT_STATIC         = YES (incluir atributos/funções estáticas)
HIDE_UNDOC_MEMBERS     = NO
HIDE_UNDOC_CLASSES    = NO

#-----
# configuration options related to the LaTeX output
#-----
GENERATE_LATEX         = YES
LATEX_OUTPUT           = (diretório opcional, por default cria diretório latex)
COMPACT_LATEX          = NO
PAPER_TYPE             = a4wide (formato da folha)
EXTRA_PACKAGES         =
LATEX_HEADER           = (link para header)
PDF_HYPERLINKS         = YES (gerar links para pdf)
USE_PDFLATEX           = YES (gerar arquivo pdf)
LATEX_BATCHMODE        = NO

#-----
# configuration options related to the RTF output
#-----
GENERATE_RTF           = NO
RTF_OUTPUT             =
COMPACT_RTF            = NO
RTF_HYPERLINKS        = NO
RTF_STYLESHEET_FILE    =
RTF_EXTENSIONS_FILE    =

#-----
# configuration options related to the man page output
#-----
GENERATE_MAN           = NO
MAN_OUTPUT             =
MAN_EXTENSION          =
MAN_LINKS              = NO

#-----
# configuration options related to the XML output
#-----
GENERATE_XML           = NO

#-----
# Configuration options related to the preprocessor
#-----
ENABLE_PREPROCESSING   = YES
MACRO_EXPANSION        = NO
EXPAND_ONLY_PREDEF     = NO
SEARCH_INCLUDES        = YES
INCLUDE_PATH           =
INCLUDE_FILE_PATTERNS =
PREDEFINED              =
EXPAND_AS_DEFINED      =

#-----
# Configuration::addtions related to external references
#-----
TAGFILES               =
GENERATE_TAGFILE        =
ALLEXTERNALS           = NO
PERL_PATH              =

#-----
# Configuration options related to the dot tool
#-----
HAVE_DOT               = YES
CLASS_GRAPH            = NO (se true, desabilita geração do grafico das heranças e gera gráfico das heranças e relações)
COLLABORATION_GRAPH    = YES (grafico da hierarquia e relacionamentos)
INCLUDE_GRAPH          = YES (grafico dos arquivos include)
INCLUDED_BY_GRAPH      = YES
GRAPHICAL_HIERARCHY    = YES (gera diagrama de heranças)
DOT_PATH               = /home/SYSTEM/gv1.7c/bin
MAX_DOT_GRAPH_WIDTH    = 1024
MAX_DOT_GRAPH_HEIGHT   = 1024
GENERATE_LEGEND         = YES
DOT_CLEANUP            = YES (deleta arquivos temporários)

#-----
# Configuration::addtions related to the search engine
#-----
```

```

SEARCHENGINE      = NO
CGI_NAME          =
CGI_URL           =
DOC_URL           =
DOC_ABSPATH       =
BIN_ABSPATH       =
EXT_DOC_PATHS     =

```

Observe na listagem acima a opção de uso do programa auxiliar “dot tool”. O mesmo é utilizado para gerar diagramas de relacionamento entre as classes.

Para obter informações gerais sobre o doxygen execute:

```
doxygen --help
```

A saída do comando (doxygen - -help) é dada por

Listing 11.2: Saída do comando doxygen - -help.

```

Doxygen version 1.2.8.1 Copyright Dimitri van Heesch 1997-2001

You can use doxygen in a number of ways:

1. Use doxygen to generate a template configuration file:
doxygen [-s] -g [configName]
If - is used for configName doxygen will write to standard output.

2. Use doxygen to update an old configuration file:
doxygen [-s] -u [configName]

3. Use doxygen to generate documentation using an existing configuration file:
doxygen [configName]
If - is used for configName doxygen will read from standard input.

4. Use doxygen to generate a template style sheet file for RTF, HTML or Latex.
RTF:
doxygen -w rtf styleSheetFile
HTML:
doxygen -w html headerFile footerFile styleSheetFile [configFile]
LaTeX:
doxygen -w latex headerFile styleSheetFile [configFile]

5. Use doxygen to generate an rtf extensions file RTF:
doxygen -e rtf extensionsFile

If -s is specified the comments in the config file will be omitted. If configName is omitted '
  Doxyfile' will be used as a default.

```

11.3.1 Exemplo de programa documentado

A listagem a seguir apresenta um exemplo de programa documentado.

Listing 11.3: Exemplo de código documentado no formato JAVA_DOC para uso com o programa doxygen.

```

#ifndef TTeste_h
#define TTeste_h

/*
=====
PROJETO:      Biblioteca LIB_LMPT
              Assunto/Ramo: TTeste...
=====

```

```

Desenvolvido por:
    Laboratorio de Meios Porosos e Propriedades Termofisicas
    [LMPT].
@author    André Duarte Bueno
@file      TTeste.h
@begin     Sat Sep 16 2000
@copyright (C) 2000 by André Duarte Bueno
@email     andre@lmpt.ufsc.br
*/

//-----
//Bibliotecas C/C++
//-----

//-----
//Bibliotecas LIB_LMPT
//-----
//#include <Base/_LIB_LMPT_CLASS.h>

/*
=====
Documentacao CLASSE: TTeste
=====
*/
/**
@short
    Classe de teste das diversas classes da LIB_LMPT.
    O objetivo é dentro da main criar e chamar TTeste
    que cria e chama as demais classes.

Assunto:      Teste da LIB_LMPT
Superclasse:  TTeste
@author       André Duarte Bueno
@version      versão...
@see          veja assunto...
*/
class TTeste
{
//-----Atributos
private:
protected:
public:

//-----Construtor
    //Construtor
    TTeste ()
    {
    };

//-----Destrutor
    //Destrutor
    virtual ~ TTeste ()
    {
    };

```

```
//-----Métodos
private:
protected:
public:

    /** Função principal, executada por main.
        Vai solicitar ao usuário o tipo de objeto a ser criado,
        criar o objeto e passar o controle
        do programa para o objeto criado */
    void Run ();
};

//-----Friend
//Declaração de Funções Friend
//ostream& operator<< (ostream& os, TTeste& obj);
//istream& operator>> (istream& is, TTeste& obj);
#endif
```

11.3.2 Exemplo de diagramas gerados pelo doxygen

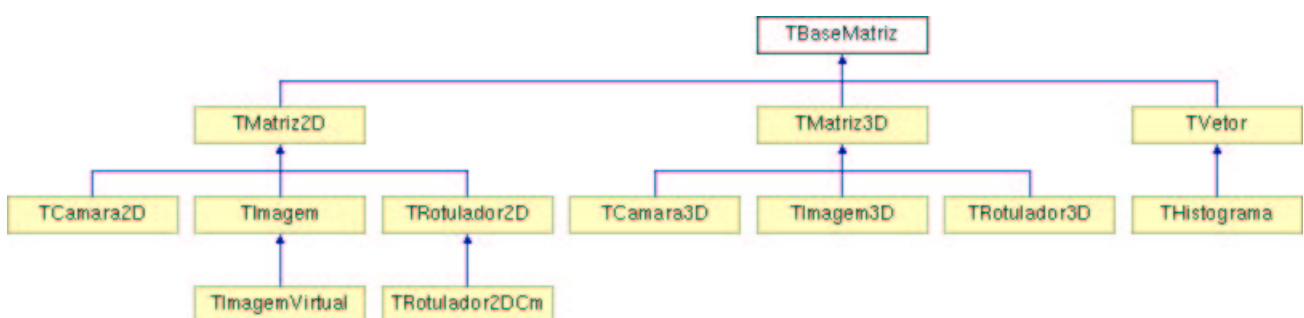
Você pode instalar em seu sistema o programa da graphvis. O programa da graphvis é utilizado para gerar diagramas das classes, ilustrando as diversas classes e seus relacionamentos. Gera ainda diagramas das dependências dos arquivos. Procure pelo programa no site (<http://www.research.att.com/sw/tools/graphviz/>).

Você pode configurar o doxygen para que use o programa da graphvis, possibilitando assim a inclusão dentro da documentação (html, tex), de Figuras ilustrando as hierarquias das diversas classes.

Para ilustrar a documentação api gerada pelo doxygen, incluí na distribuição desta apostila o arquivo LIB_LMPT-api.tar.gz. Você pode descompactar este arquivo e ver como fica a documentação gerada com o seu browser (netscape).

A título ilustrativo, apresenta-se na Figura ?? a hierarquia TMatriz da biblioteca LIB_LMPT.

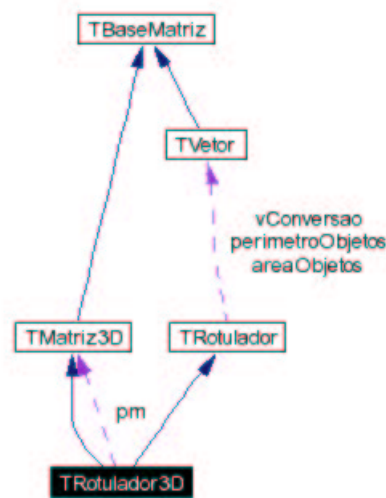
Figura 11.1: Ilustração da hierarquia TMatriz da biblioteca LIB_LMPT.



Apresenta-se na Figura ?? a hierarquia da classe TRotulador3D. Observe a relação da classe TRotulador3D com as demais classes.

Apresenta-se na Figura ?? as dependências do arquivo TRotulador3D.

Figura 11.2: Ilustração da hierarquia da classe TRotulador3D da biblioteca LIB_LMPT.



11.4 Documentação profissional com sgml/xml (LyX)

Vimos que a documentação de código é um tipo de documentação que é incluída dentro do código, utilizando o formato JAVA_DOC. Que o programa doxygen é utilizado para gerar diversos arquivos html que incluem a documentação de cada arquivo da biblioteca ou programa, e que com o doxygen pode-se gerar saída em outros formatos (como pdf, latex, rtf, xml, manpage).

Mas você também vai gerar um manual do usuário e talvez um manual científico. Neste caso como devo proceder para criar estes manuais ?

No mundo GNU/Linux, utiliza-se o DOCBOK.

O docbook é um “document type definition -DTD”, uma especificação de formato de documento. Você pode implementar o docbook usando sgml (standardized general markup language) ou xml (extensible markup language).

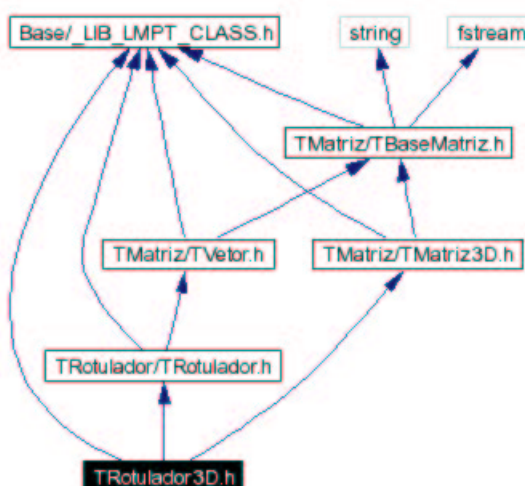
É mais ou menos assim: Com docbook você define o formato do manual, com sgml/xml você define a formatação de cada parágrafo.

Se você instalou todos os pacotes de desenvolvimento de sua distribuição, provavelmente já tem disponibilizados os pacotes necessários. No meu sistema tenho instalados os pacotes:

```

[andre@mercurio Cap4-STL]$
$rpm -qa | egrep docb && rpm -qa | egrep jade
docbook-dtd41-xml-1.0-7
docbook-utils-0.6.9-2
docbook-dtd30-sgml-1.0-10
docbook-dtd41-sgml-1.0-10
docbook-utils-pdf-0.6.9-2
docbook-dtd412-xml-1.0-1
docbook-dtd40-sgml-1.0-11
docbook-dtd31-sgml-1.0-10
docbook-style-dsssl-1.64-3
  
```

Figura 11.3: Ilustração das dependências do arquivo TRotulador3D.



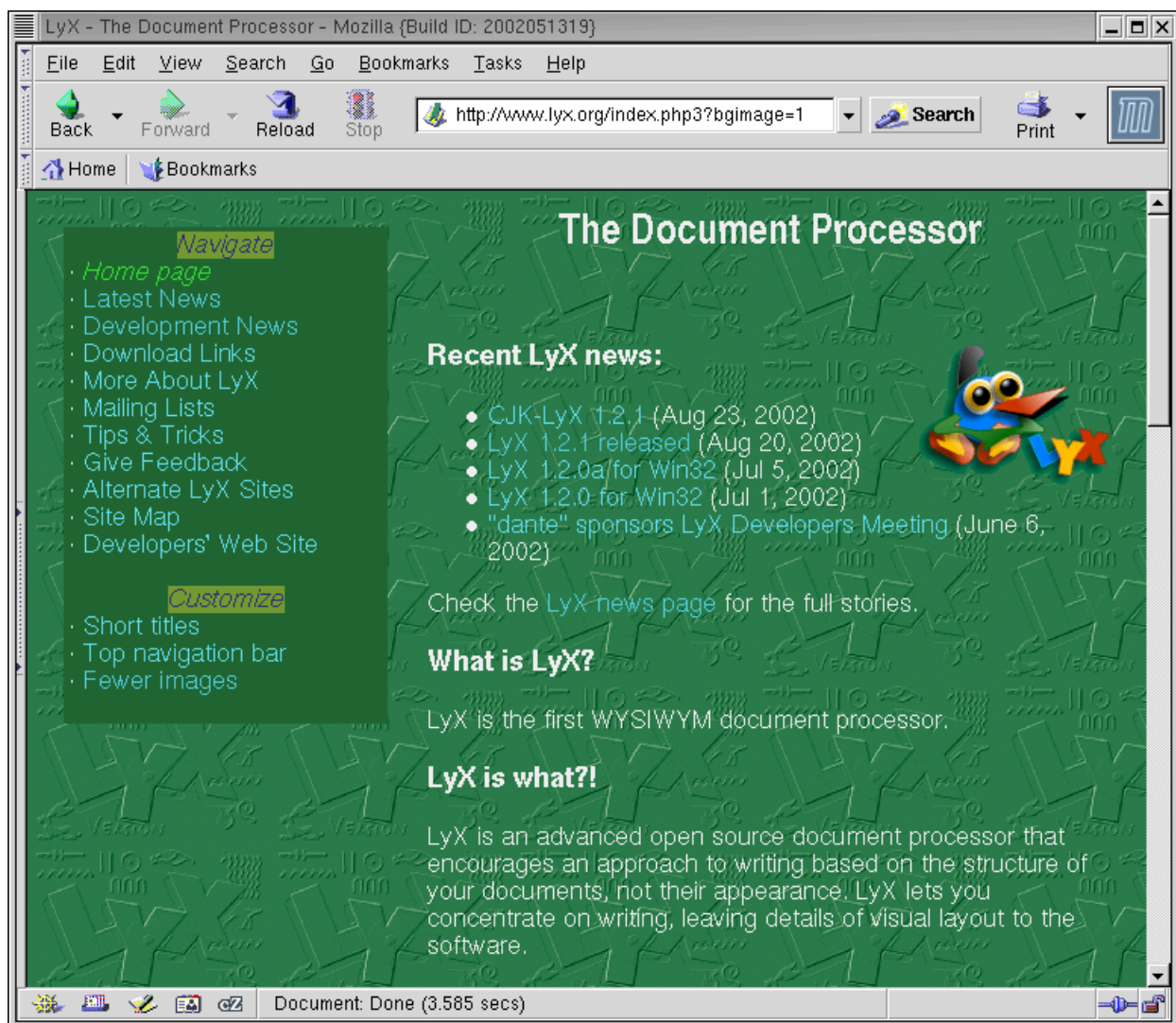
```
jadetex-3.11-4  
openjade-1.3-17
```

Veja detalhes do docbook, sgml e xml no site (<http://www.xml.org>).

Apresenta-se na Figura ?? o site do programa L^AT_EX. O programa L^AT_EX¹, tem total suporte aos formatos docbook. Desta forma você pode gerar o manual do usuário em um programa muito simples de usar (o L^AT_EX) e no final pode gerar versões do seu manual nos formatos html, pdf, sgml, txt.

¹Repito, que o L^AT_EX () é utilizado para gerar esta apostila.

Figura 11.4: O site do LyX.



Capítulo 12

Seqüência de Montagem de Um Programa GNU²

Neste capítulo vamos apresentar o desenvolvimento de programas multiplataforma utilizando as ferramentas da GNU.

12.1 Introdução a programação multiplataforma com GNU

Um projeto comum pode ser montado apenas com os arquivos de código (*.h,*.cpp) e o arquivo Makefile. O problema deste formato é a existência de centenas de Unix, GNU/Linux, sendo, cada um destes, um pouco diferentes entre si. Por exemplo, o compilador pode se chamar CC, gcc, c++, g++. Na prática isto implica na necessidade de se corrigir os arquivos Makefile para cada máquina alvo.

Para solucionar este problema, foram desenvolvidos pela GNU, um conjunto de programas que automatizam a geração de código, para as mais variadas plataformas. Dentre estes programas, os mais usuais são o aclocal, o autoheader, o automake, o autoconf e o libtool¹. Estes programas são brevemente descritos neste capítulo.

Observe que ao desenvolver um programa utilizando as ferramentas multiplataforma da GNU, você poderá compilar seu código em qualquer máquina Unix, GNU/Linux, Aix, Solaris, MacOS X, Windows. Pois um projeto GNU inclui o ./configure, um script de shell, que cria os arquivos Makefile para cada máquina.

Observe na Tabela ??, o diagrama das tarefas executadas pelo programador².

O primeiro programa a ser executado é o aclocal, o mesmo recebe como entrada um conjunto de arquivos de código e um arquivo configure.in, gerando os arquivos aclocal.m4 e acsite.m4.

A seguir, executa-se o ifnames para identificação dos includes e defines em comum. O programador usa o autoscan para gerar um esqueleto inicial do arquivo configure.scan, a seguir, o programador usa um editor como o emacs para modificar o arquivo configure.scan, gerando o configure.in. O autoheader é usado para gerar o arquivo config.h.

O programador deve gerar diversos arquivos makefile.am, um para cada diretório e então executar o automake. O automake converte os arquivos makefile.am em makefile, podendo os mesmos ser executados com o programa make.

¹descrito no Capítulo ??.

²O ambiente de desenvolvimento do kdevelop tem um “wizard” que gera automaticamente todos estes arquivos para você.

Finalmente, o programador executa o programa autoconf. O autoconf gera um script de configuração do programa, usando para isto o arquivo configure.in. A saída do autoconf é o arquivo configure, o mesmo será executado pelo usuário para gerar os makefiles específicos para a máquina do usuário. Veja na Tabela ?? a sequência executada pelo usuário.

Tabela 12.1: Sequência executada pelo programador para montagem de programas multi-plataforma usando o padrão GNU.

	Entrada.	Programa executado	Saída.
1	*.h, *.cpp	aclocal*	aclocal.m4
	configure.in		acsite.m4
2	*.h*, *.cpp	ifnames*	Lista dos defines dos diversos arquivos
3	*.h, *.cpp	autoscan*	configure.scan
4	configure.scan	usuário	configure.in
5	configure.in config.h.top acconfig.h config.h.bot	autoheader*	acsite.m4 config.h.in
6		usuário	Makefile.am
7	Makefile.am	automake*	Makefile.in
8	aclocal.m4 acsite.m4 configure.in	autoconf*	configure

Apresenta-se na Tabela ?? a sequência de tarefas executadas pelo usuário. Esta sequência é executada na máquina do usuário para gerar o programa para a sua máquina. Observe que uma das grandes vantagens deste tipo de distribuição, é que se o usuário tem uma máquina Pentium IV, o compilador vai receber instruções para compilar o programa para um Pentium IV e não para um antigo 386.

Tabela 12.2: Sequência executada pelo usuário.

Entrada	Programa executado	Saída
Makefile.in	./configure	Makefile config.cache config.log
config.h.in Makefile.in	config.status*	config.h Makefile
Makefile	make	Código Compilado
Código compilado	make install	Programa Instalado

12.2 aclocal

O programa aclocal gera o arquivo aclocal.m4. O aclocal procura todos os arquivos *.m4 na path do automake/autoconf, depois procura o arquivo configure.in, copiando todas as macros encontradas para o arquivo aclocal.m4. Ou seja, todas as macros serão copiadas para o arquivo aclocal.m4.

Protótipo e parâmetros do aclocal:

aclocal [opções]

- *-help* *Help do aclocal.*
- *-acdir=dir* *Define o diretório.*
- *-output=file* *Define nome do arquivo de saída.*
- *-verbose* *Modo verbose (detalhado).*
- *-version* *Mostra versão do aclocal.*

12.3 ifnames

O programa ifnames pesquisa toda a estrutura de diretórios e lista todos os defines dos arquivos *.h e *.cpp. O programa ifnames agrupa todos os defines em um único arquivo o que é útil para gerar o arquivo config.h.in

Protótipo do ifnames:

ifnames [-h][- -help][-mdir][--macrodir=dir][- -version][file...]

- *-help[-h]* *Mostra help.*
- *-verbose[-v]* *Modo verbose.*
- *-version* *Mostra versão.*

12.4 autoscan

O autoscan pesquisa a estrutura de diretórios e busca arquivos *.h e *.cpp, gerando o arquivo configure.scan. O autoscan extrai informações do código e dos headers, como chamadas de funções.

Protótipo e parâmetros do autoscan:

autoscan [- -macrodir=dir][- -help][- -verbose]

- *-help* *Mostra help.*
- *-verbose[-v]* *Modo verbose.*
- *-version* *Mostra versão.*
- *-srcdir* *Diretório a ser escaneado.*

12.4.1 Roteiro do autoscan

1. Execute o autoscan para gerar o configure.scan. O arquivo configure.scan serve de esboço inicial para o arquivo configure.in
2. Corrija o arquivo configure.scan, incluindo ali as macros e definições necessárias.
3. Renomeie o arquivo configure.scan para configure.in. Observe que o arquivo configure.in é um gabarito usado para gerar o arquivo configure final.

12.5 autoheader

O autoheader pode ser usado para gerar o arquivo config.h.in. O arquivo config.h.in é usado pela macro AC_CONFIG_HEADER(file) para gerar o arquivo config.h. O arquivo config.h contém definições compartilhadas por todos os arquivos do pacote (é um header comum a todo pacote).

Protótipo e parâmetros do autoheader:

autoheader

- *-help[-h]* Mostra help.
- *-localdir=dir[-l dir]*
- *-macrodir=dir[-m dir]*
- *-version* Mostra versão.

12.5.1 Roteiro do autoheader

1. Crie um arquivo config.top com as instruções iniciais do config.h.in.
2. Criar o arquivo acconfig.h

```
/* Define if the C++ compiler supports BOOL */
#undef HAVE_BOOL
#undef VERSION
#undef PACKAGE
/* Define if you need the GNU extensions to compile */
#undef _GNU_SOURCE
```

3. Crie um arquivo config.bot com as instruções finais do config.h.in.
4. Execute o autoheader.
5. Edite o arquivo config.h.in gerado pelo autoheader.
6. Inclua no arquivo configure.in a macro AC_CONFIG_HEADER().

12.6 automake

12.6.1 Introdução ao automake

O automake é uma ferramenta para automatizar a criação de makefiles independentes de plataforma. Basicamente o programador escreve um arquivo Makefile.am para cada diretório, o automake lê estes arquivos e o arquivo configure.in e cria os arquivos Makefile.in. O arquivo Makefile.in é um gabarito para geração do Makefile final, isto é, os arquivos Makefile.in serão usados pelo ./configure para geração dos arquivos Makefile finais. A grande vantagem é não precisar reescrever os arquivos Makefile para cada máquina em que você vai compilar o programa.

Vimos no capítulo Make como criar um arquivo makefile simples. O automake cria arquivos makefile complexos.

O automake suporta três tipos de estruturas de diretório

flat	Todos os arquivos estão em um único diretório.
deep	Existem vários subdiretórios e arquivos de configuração no diretório base. Dentro do Makefile.am existe a macro SUBDIRS.
shallow	O código primário está no diretório de mais alto nível e os outros códigos em subdiretórios (usado em bibliotecas).

O automake suporta três tipos de opções de verificação

'foreign'	Checa somente o necessário (adotado pelo kdevelop).
'gnu'	É o default, verifica a presença dos arquivos padrões da gnu (INSTALL, NEWS, README, COPYING, AUTHORS, Changelog)
'gnits'	É um padrão que verifica arquivos adicionais ao padrão gnu. É o mais extenso.

Padronização do formato dos nomes

Os nomes das macros do automake obedecem um padrão uniforme. Se o seu pacote tem um programa chamado meu-programa.1, o mesmo será tratado como meu_programa_1, ou seja substitui - por _ e . por _ .

Protótipo e parâmetros do automake:

automake [opções]

-a [-add-missing]

-amdir=dir Define diretório.

-help Ajuda / Help.

-generate-deps Gera dependências.

-output-dir=dir Diretório de output/saída.

-srcdir=dir Diretório de fonte.

- `-v`[`-verbose`] *Modo verbose (detalhado).*

- Como é um arquivo makefile, existem instruções como:
`PROGRAMS`= lista dos programas a serem compilados.
`EXTRA_PROGRAMS`= programas adicionais.
`bin_PROGRAMS` = programas binários.
`sbin_PROGRAMS` = `@PROGRAMS@`.

Exemplo:

```
bin_PROGRAMS = nomeDoPrograma
nomeDoPrograma_SOURCES = nome.h nome.cpp ...
nomeDoPrograma_LDADD = @LIBOBJJS@
```

Dica: A variável `@LIBOBJJS@` é copiada do arquivo `configure.in` para o arquivo makefile (substituição simples). Observe que o valor da variável `@LIBOBJJS@` é definida quando o usuário executa o `./configure`.

Roteiro do automake

Apresenta-se a seguir o roteiro de execução do automake em um projeto simples.

1. Edite o `configure.in`, acrescentando as macros:

```
AM_INIT_AUTOMAKE(nome_programa,versao)
AC_REPLACE_FUNCS
LIBOBJJS=listaobjetos da biblioteca
AC_SUBST(LIBOBJJS)
```

2. Cria e edita o arquivo `makefile.am`

```
SUBDIRS = lib_lmpt
EXTRA_DIST = AUTHORS COPYING ChangeLog INSTALL README TODO ORGANIZATION HOWTO
AUTOMAKE_OPTIONS = foreign
bin_PROGRAMS= nomePrograma
nomePrograma= lista arquivos cpp
nomePrograma_LDADD= @LIBOBJJS@
```

3. Executa o automake.

Macros do automake

Uma macro realiza um conjunto de operações. Apresenta-se a seguir um conjunto de macros que podem ser incluídas nos arquivos `configure.in` e `makefile.am`. Estas macros são usadas para interfacear o automake com o autoconf. As macros do autoconf iniciam com `AC` e as do automake com `AM`.

AC_CONFIG_HEADER

O automake requer a macro `AM_CONFIG_HEADER` que é similar a `AC_CONFIG_HEADER`.

AC_PATH_XTRA

Insera as definições do AC_PATH_XTRA em cada arquivo Makefile.am.

LIBOBJS

Inclui os arquivos *.o na lista do LIBOBJS.

AC_PROG_RANLIB

Necessário se o pacote compila alguma biblioteca.

AC_PROG_CXX

Necessário se existe código em C++.

AM_PROG_LIBTOOL

Executa o pacote libtool.

ALL_LINGUAS

Checa o diretório po e os arquivos .po para especificações relativas a linguagens.

AUTOMAKE_OPTIONS

Variável especial definida nos arquivos Makefile.am. Suporta um subconjunto de variáveis que realizam tarefas específicas.

Exemplos:

`dist-tarZ` Cria uma distribuição do pacote no formato `.tar.gz`.

Apresenta-se a seguir um conjunto de macros que podem ser incluídas no arquivo `configure.am` e que são fornecidas pelo automake. Estas macros são usadas para interfacear o automake com o autoconf.

AM_INIT_AUTOMAKE(nomePacote,versão)

Inicializa o automake, rodando um conjunto de macros necessárias ao `configure.in`

AM_CONFIG_HEADER

Reconfigurar o arquivo `config.h` levando em conta parâmetros do automake

AM_ENABLE_MULTILIB**AM_FUNC_MKTIME****AM_PROG_CC_STDC**

Se o compilador não está em ANSIC tenta incluir parâmetros para deixá-lo no formato ANSIC

AM_SANITY_CHECK

Verifica se os arquivos *.o criados são mais novos que os de código

A macro AM_INIT_AUTOMAKE

Existe um conjunto de tarefas que precisam ser realizadas, e que são realizadas automaticamente pela macro AM_INIT_AUTOMAKE. As mesmas podem ser executadas manualmente por uma sequência da forma:

Definição das variáveis PACKAGE e VERSION

Uso da macro AC_ARG_PROGRAM

Uso da macro AM_SANITY_CHECK

Uso da macro AC_PROG_INSTALL

12.6.2 Sentenças para o automake

- Leia o manual do automake, a parte principal, com exemplos não é extensa.
- Qualquer variável ou macro definida em um arquivo makefile.am sobrescreve variáveis definidas no configure.in.
- Você pode incluir variáveis do sistema em seu arquivo Makefile.am. Desta forma, pode passar parâmetros para diversos Makefile.am de forma simplificada.

– Exemplo:

```
CXXFLAGS = -I${PATH_LIB_LMPT_INCLUDES}  
-I${PATH_LIB_IMAGO} -I${PATH_LIB_COILIB}  
DEFS = -D__LINUX__ -D__INTEL__ -D__X11__ -D__MESA__
```

12.7 autoconf

12.7.1 Introdução ao autoconf

O autoconf é uma ferramenta que objetiva automatizar a configuração de seu software para a plataforma alvo.

O autoconf inicializa pesquisando as macros instaladas com o pacote autoconf, a seguir verifica a presença do arquivo opcional acsite.m4 (no diretório de instalação do autoconf) e pelo arquivo aclocal.m4 (no diretório do programa). O arquivo aclocal.m4 é criado pelo aclocal.

O resultado da execução do autoconf é a criação do arquivo configure. O arquivo configure será executado pelos usuários para geração dos arquivos Makefile, adaptados a plataforma onde o programa vai ser compilado³.

Dica de portabilidade: Se for usar uma biblioteca não portátil, procure criar um módulo separado.

³Lembre-se, quando você faz o download de um programa no formato .tar.gz, você primeiro descompacta o arquivo e a seguir executa: ./configure && make && make install. Observe que quando você executa o ./configure, que realiza uma série de verificações em seu sistema e cria os arquivos makefiles finais.

12.7.2 Protótipo do autoconf

Protótipo e parâmetros do autoconf:

autoconf

-help[-h] Mostra um help.

-localdir=dir[-l dir] Define diretório local.

-macrodir=dir[-m dir] Define diretório de macros.

-version Mostra versão.

Vamos apresentar a seguir um exemplo de arquivo `configure.in` e depois descrever rapidamente as macros que você pode incluir no `configure.in`.

12.7.3 Roteiro do autoconf

1. Edite o `configure.in`, acrescentando as macros:

```
AM_INIT_AUTOMAKE (nome_programa, versao)
AC_REPLACE_FUNCS
LIBBOJS=listaobjetos da biblioteca
AC_SUBST (LIBBOJS)
```

12.7.4 Estrutura de um arquivo `configure.in`

Como visto, você pode utilizar o `autoscan` para gerar um esboço inicial do arquivo `configure.in`.

O arquivo `configure.in` é usado pelo `autoconf` para montagem do programa shell `configure`. O `configure.in` é composto de um conjunto de macros que serão usadas para testar a configuração de seu micro (qual compilador está instalado, sua localização,...).

Basicamente o programador escreve um arquivo `configure.in` contendo:

```
Inclue as macros de inicialização AC_INIT
Inclue as macros de testes
checks for prograns
checks for libraries
checks for headers files
checks for typedefs
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services
Inclue as macros de finalização AC_OUTPUT
```

12.7.5 Exemplo de um arquivo configure.in

Apresenta-se a seguir o exemplo de um arquivo configure.in. Observe que uma linha *dnl* é uma linha de comentário.

O nome do arquivo dentro do AC_INIT(Makefile.am) é usado apenas para verificar o diretório. Os arquivos gerados pelo autoconf estão em AC_OUTPUT. Observe neste exemplo que dentro da macro AC_OUTPUT estão listados os diversos arquivos makefile a serem criados.

```
dnl Exemplo de arquivo configure.in
dnl linha de comentário
dnl Process this file with autoconf to produce a configure script.
AC_INIT(Makefile.am)
AM_CONFIG_HEADER(config.h)
AM_INIT_AUTOMAKE(lib_lmpt,0.1)
dnl Checks for programs.
AC_PROG_CC
AC_PROG_CXX
AC_PROG_RANLIB
dnl Checks for libraries.
dnl Checks for header files.
dnl Checks for typedefs, structures, and compiler characteristics.
dnl Checks for library functions.
AC_OUTPUT(Makefile lib_lmpt/Makefile lib_lmpt/include/Base/Makefile
lib_lmpt/include/Makefile lib_lmpt/source/Base/Makefile lib_lmpt/source/Makefile)
```

Dica: Não deixar espaços entre o nome da macro e os paranteses.

12.7.6 Macros do autoconf

Lista-se a seguir um conjunto de macros que podem ser utilizadas para testar a presença de algo. Estas macros fazem parte do autoconf e devem ser incluídas no arquivo configure.in. Observe que iniciam com AC se forem macros do autoconf e AM se forem macros do automake.

As principais macros a serem incluídas no arquivo configure.in são a AC_INIT e AC_OUTPUT.

AC_INIT(arquivo)

- Processa os parâmetros da linha de comando, e
- Pesquisa a estrutura de diretórios pelos arquivos *.h e *.cpp.
- O nome do arquivo dentro do AC_INIT é usado apenas para verificar se o diretório está correto.

AC_OUTPUT([arquivo],comandos extras,[comandos inicialização]))

- Macro que gera os arquivos de saída do comando autoconf. Gera os arquivos Makefile.in e configure. Observe que você pode passar comandos extras e comandos de inicialização. Ou seja, no AC_OUTPUT serão colocados os nomes dos arquivos que serão gerados.

Mas existem muitas outras macros que estão disponíveis e que você pode usar, macros para testes em geral, para pesquisar se determinado programa/biblioteca esta instalado(a), se determinadas funções estão disponíveis, entre outros. Lista-se a seguir as macros mais utilizadas.

Testes de uso geral

AC_CONFIG_AUX_DIR

Configurações auxiliares.

AC_OUTPUT_COMMANDS

Execução de comandos de shell adicionais.

AC_PROG_MAKE_SET

Usado para definir a variável MAKE=make.

AC_CONFIG_SUBDIRS

Rodar o configure em cada um dos subdiretórios incluídos nesta listagem.

AC_PREFIX_DEFAULT(prefix)

Seta o prefixo default para instalação (o padrão é /usr/local).

AC_PREFIX_PROGRAM(program)

Se o usuário não entrar com -prefix, procura por um prefixo na PATH

AC_PREREQ(version)

Informa a versão do autoconf que deve ser utilizada, se a versão instalada for anterior emite mensagem de erro.

AC_REVISION(revision-info)

Inclue mensagem no início dos arquivos informando o número da revisão.

Pesquisando programas

AC_PROG_CPP

Seta a variável CPP com o nome do pré-processador C existente.

AC_PROG_CXX

Verifica se já foi definida a variável CXX ou CCC (nesta ordem). Se não definida procura o compilador C++ e seta a variável CXX. Se o compilador for da GNU, seta a variável GXX=yes. Se CXXFLAGS não foi definido seta como -g -O2.

AC_PROG_CC

Identifica o compilador C, e define a variável CC com o nome do compilador encontrado. Adicionalmente se encontrou o GNU-GCC define a variável GCC=yes caso contrário GCC=.

AC_PROG_CXXCPP

Seta a variável CXXCPP com o nome do pré-processador C++.

AC_PROG_INSTALL

Seta a variável INSTALL com a path compatível com o programa de instalação BSD.

AC_PROG_LN_S

Verifica se o sistema aceita links simbólicos e seta a variável LNS como ln -s.

AC_PROG_RANLIB

Seta a variável RANLIB se o ranlib esta presente.

AC_CHECK_PROG(variável, programa, açãoTrue, açãoFalse)

Checa a existência da variável e do programa, se ok realiza ação true se false realiza ação false.

AC_CHECK_PROGS(variável, programas, açãoTrue, açãoFalse)

Checa a existência da variável e dos programas, se ok realiza ação true se false realiza ação false.

Pesquisando bibliotecas**AC_CHECK_LIB(biblioteca, função, açãoTrue, açãoFalse)**

Verifica se a função pertence a biblioteca.

AC_HAVE_LIBRARY(biblioteca, açãoTrue, açãoFalse)

Verifica se a biblioteca existe.

AC_SEARCH_LIB(função, listaDeBibliotecas, açãoTrue, açãoFalse)

Pesquisa a função no conjunto de bibliotecas listadas.

AC_TRY_LINK**AC_TRY_LINK_FUNC****AC_COMPILE_CHECK****Pesquisando funções****AC_CHECK_FUNC(função, açãoTrue, açãoFalse)**

Verifica se a função existe, e executa o comando de shell.

AC_CHECK_FUNCS(função..., açãoTrue, açãoFalse)

Verifica se a função existe, e executa o comando de shell.

AC_REPLACE_FUNCS(função...)

Adiciona a função com o nome função.o a variável LIBOBJS.

Pesquisando arquivos *.h e *.cpp**AC_CHECK_HEADER(header, açãoTrue, açãoFalse)**

Verifica a existência do header, se existe executa açãoTrue.

AC_CONFIG_HEADER(header_a_ser_criado)

Arquivo a ser criado com os #defines. Substitui @DEFS@ por -DHAVE_CONFIG_H, o nome padrão para o arquivo é config.h. Usado para criar o arquivo config.h com os header comuns ao pacote.

AC_CHECK_FILE(arquivo,açãoTrue,açãoFalse)

Checa se o arquivo existe.

AC_CHECK_FILES(arquivos,açãoTrue,açãoFalse)

Checa um conjunto de arquivos.

AC_TRY_CPP(includes[,açõesTrue,açõesFalse])

Procura pelos arquivos include, se localizou realiza a ação true, caso contrário a ação false.

AC_EGREP_HEADER(padrãoPesquisa,headerFile,ação)

Se a pesquisa do padrão no arquivo header foi ok, realiza a ação.

AC_EGREP_CPP(padrãoPesquisa,cppFile,ação)

Se a pesquisa foi ok, realiza a ação.

AC_TRY_COMPILE(includes,corpoDaFunção,açãoTrue,açãoFalse)

Cria um programa de teste, com a função especificada para verificar se a função existe.

PS: Arquivos de headers especificados pelas macros HEADERS..., geralmente não são instalados, e os headers listados em ..._SOURCES não podem ser incluídos nos ..._HEADERS.

Rodando programas de teste**AC_TRY_RUN(programa,açãoTrue,açãoFalse)**

Tenta rodar o programa, se ok realiza açãoTrue.

Pesquisando estruturas: Veja manual do autoconf.

Pesquisando typedefs: Veja manual do autoconf.

Pesquisando características do compilador C**AC_C_CONST**

Verifica se o compilador suporta variáveis const.

AC_C_INLINE

Verifica se o compilador suporta funções inline.

AC_CHECK_SIZEOF(tipo[,tamanho])

Ex: AC_CHECK_SIZEOF(int *).

12.7.7 Como aproveitar os resultados das pesquisas realizadas pelo autoconf

Você coloca um conjunto de macros no arquivo configure.in para testar o seu programa. Os resultados das pesquisas realizadas pelo autoconf podem ser salvos. Como exemplos, definições de diretrizes de pré-processador, definição de variáveis de shell. Você pode salvar os resultados em caches, ou imprimir os resultados das pesquisas na tela. As macros abaixo mostram como fazer:

AC_DEFINE(variável,valor,descrição)

Cria variável define. Ex: AC_DEFINE(EQUATION, \$a > \$b)

AC_SUBST(variável)

Cria uma variável de saída a partir de variável de shell.

Define o nome de uma variável que deverá ser substituída nos arquivos Makefile.

AC_SUBST_FILE(variável)

O mesmo que acima.

AC_CACHE_VAL(cache-id,comando_do_id)

Veja manual do autoconf.

AC_CACHE_CHECK

Verifica o cache.

AC_CACHE_LOAD

Carrega algo do cache.

AC_CACHE_SAVE

Salva algo no cache.

AC_MSG_CHECKING(descrição)

Informa o usuário que executou o ./configure o que esta fazendo.

AC_MSG_RESULT(descrição_do_resultado)

Normalmente uma mensagem com o resultado do que foi checado.

AC_MSG_ERROR(descrição_da_mensagem_de_erro)

Emite uma mensagem de erro.

AC_MSG_WARN(descrição_da_mensagem_de_warning)

Emite uma mensagem de warning.

Como definir variáveis no configure.in e usar no makefile.am

Apresenta-se a seguir um exemplo de definição de variável no arquivo configure.in que vai ser usada no arquivo makefile.am.

No arquivo configure.in

```
SUBDIRS= "source doc"  
AC_SUBST(SUBDIRS)
```

No arquivo makefile.am

```
##linha de comentário  
VARIABEL=valor  
PROG_SOURCES= prog.cpp $(VARIABEL)  
SUBDIRS= @SUBDIRS@
```

12.7.8 Variáveis definidas no arquivo configure.in e que serão substituídas no arquivo Makefile

Além de testar o sistema como um todo, o autoconf permite a inclusão de definições no arquivo configure.in que serão substituídas nos arquivos Makefile. Desta forma, você não precisa ficar conferindo se uma variável definida em cada arquivo Makefile.am esta coerente. Basta definir a variável no configure.in, que a mesma será copiada para cada arquivo Makefile. Apresenta-se a seguir uma lista resumida das variáveis que são definidas no configure.in para uso do makefile.am.

bindir Diretório de instalação do executável.

configure_input Para incluir um comentário informando que o arquivo foi gerado pelo autoconf.

datadir Diretório para instalação dos arquivos ready-only.

exec_prefix Prefixo dos arquivos executáveis que dependem da arquitetura.

includedir Diretório para instalação dos arquivos de headers.

infodir Diretório para instalação dos arquivos de documentação.

libdir Diretório para instalação dos objetos das bibliotecas.

libexecdir Diretório para instalação de executáveis que outros programas executam.

prefix Prefixo para arquivos dependentes de plataforma.

CXXCOMPILE

Comando para compilação de códigos C++, normalmente setado como:

```
CXXCOMPILE= $(CXX) $(DEFS) $(INCLUDES)
$(AM_CPPFLAGS) $(CPPFLAGS) $(AM_CXXFLAGS) $(CXXFLAGS)
```

CXX

Nome do compilador C++.

DEFS

Opção -D para compiladores C.

CFLAGS

Opções de debugagem e otimização para compiladores C.

CPPFLAGS

Diretório com arquivos headers e outros parâmetros para o pré-processador e compilador C.

CXXFLAGS

Opções de debugagem e otimização para compiladores C++.

CXXLINK

Comando para linkagem de programas C++, normalmente setado como:

```
CXXLINK=$(CXXLD) $(AM_CXXFLAGS) $(CXXFLAGS) $(LDFLAGS)
```

LDFLAGS

Opções para o linker.

LIBS

Opções -l e -L para o linker.

Você pode incluir em seus arquivos makefile.am variáveis definidas no shell. Ou seja, antes de executar o `./configure && make && make install`, o usuário define algumas variáveis em seu shell (ex: `PATH_INCLUDE_LIB_LMPT=/usr/include/lib_lmpt`).

Escrevendo suas macros para utilização com o autoconf

Apresentou-se acima um conjunto de macros disponíveis no pacote autoconf para uso no arquivo `autoconf.in`. Adicionalmente, você pode construir suas próprias macros e incluir no arquivo `configure.in`. Veja o manual do autoconf.

Variáveis relacionadas ao autoconf

Veja o manual do autoconf.

Variáveis de ambiente setadas pelo autoconf

Veja o manual do autoconf.

Variáveis geradas pelo autoconf

Veja o manual do autoconf.

Defines usados pelo autoconf (e que você não deve usar nos seus programas)

Veja o manual do autoconf.

Lista das macros do autoconf (macros que podem ser incluídas no `configure.in`)

Veja o manual do autoconf.

Dica: Observe que no arquivo `configure.in` existem macros do autoconf (iniciadas com `AC_`) e do automake (iniciados com `AM_`), mostrando uma interdependência do automake e do autoconf.

12.8 autoreconf

Pacote utilizado para reconstruir aquilo que o autoconf construiu, no caso de alterações na instalação.

O autoreconf atualiza os scripts `configure`.

Protótipo e parâmetros do autoreconf:

autoreconf

-help[-h] Mostra help.
-force
-localdir=dir[-l dir] Define diretório.
-macrodir=dir[-m dir]
-verbose[-v] Modo verbose.
-version Mostra versão.

12.9 ./configure

A execução do ./configure gera:
Um conjunto de arquivos Makefile.
Um arquivo de header com defines.
Um arquivo config.status.
Um arquivo de shell que salva os dados de configuração em um cache.
Um arquivo config.log com as informações da execução do ./configure.

12.10 Como incluir instruções do libtool em seu pacote gnu

Você precisa incluir os arquivos:

- 'config.guess' Nomes canônicos.
- 'config.sub' Validação de nomes.
- 'ltmain.sh' Implementa funções básicas do libtool.

Você pode usar o programa libtoolize. O programa libtoolize adiciona aos seus arquivos as instruções para dar suporte ao libtool, adicionando os arquivos 'config.guess' 'config.sub' 'ltmain.sh'.

Protótipo e parâmetros do libtoolize:

libtoolize [opções]

-automake
-copy Copia os arquivos e não os links para o diretório.
-n Não modifica os arquivos, apenas mostra as modificações.
-force Força a substituição dos arquivos existentes do libtool.
-help Ajuda.

12.10.1 Exemplo de arquivo makefile.am usando o libtool

Exemplo:

```
bin_PROGRANS = prog prog.debug
#Gera o programa
prog_SOURCES = *.cpp
prog_LDADD= libNome.a      ##"-dlopen"
#Gera o programa com debug
prog_debug_SOURCES = *.cpp
prog_debug_LDADD= libNome.a      ##"-dlopen"
prog_debug_LDFLAGS= -static
```

12.10.2 Exemplo de arquivo configure.in usando o libtool

Exemplo, acrescentar as macros:

```
##suporte do autoconf ao libtool
AC_PROG_LIBTOOL
##suporte do automake ao libtool
AM_PROG_LIBTOOL
##suporte a bibliotecas dinâmicas (?)
AC_LIBTOOL_DLOPEN
```

Sentenças:

- Leia os capítulos “Using Libttol” e “integrating libttol” do manual do libtool.
- Basicamente o libtool é suportado pela variável LTLIBRARIES.

12.11 Exemplo Completo

Apresenta-se a seguir os arquivos do programa LIB_LMPT. A estrutura de diretórios é da forma

```
LIB_LMPT
LIB_LMPT/lib_lmpt (Arquivos main.cpp, teste.cpp, teste.h)
LIB_LMPT/lib_lmpt/source/base (Arquivos TOperacao.cpp, TMath.cpp)
LIB_LMPT/lib_lmpt/include/base (Arquivos TOperacao.h, TMath.h)
LIB_LMPT/lib_lmpt/docs
```

LIB_LMPT/Makefile.am

```
SUBDIRS = lib_lmpt
EXTRA_DIST = AUTHORS COPYING ChangeLog INSTALL README TODO ORGANIZATION HOWTO
AUTOMAKE_OPTIONS = foreign
```

LIB_LMPT/Madefile.dist

```
default: all
dist: @echo "This file is to make it easier for you to create all you need"
aclocal
autoheader
# use --include-deps, if you want to release the stuff. Don't use it for yourself
automake --include-deps
autoconf
touch stamp-h.in
LIST=`find ./po -name "*.po"`; \
for i in $$LIST; do \
file2=`echo $$i | sed -e "s#\.po#\gmo#"`; \
msgfmt -o $$file2 $$i; \
done
rm -f Makefile.dist
all: aclocal autoheader automake autoconf
```

LIB_LMPT/acconfig.h

```
/* Define if the C++ compiler supports BOOL */
#undef HAVE_BOOL
#undef VERSION
#undef PACKAGE
/* Define if you need the GNU extensions to compile */
#undef _GNU_SOURCE
```

LIB_LMPT/aclocal.m4

Arquivo grande contendo um conjunto de macros.

LIB_LMPT/config.cache

```
# This file is a shell script that caches the results of configure
# tests run on this system so they can be shared between configure
# scripts and configure runs. It is not useful on other systems.
# If it contains results you don't want to keep, you may remove or edit it.
#
# By default, configure uses ./config.cache as the cache file,
# creating it if it does not exist already. You can give configure
# the --cache-file=FILE option to use a different cache file; that is
# what configure does when it calls configure scripts in
# subdirectories, so they share the cache.
# Giving --cache-file=/dev/null disables caching, for debugging configure.
# config.status only pays attention to the cache file if you give it the
# --recheck option to rerun configure.
#
```

```
ac_cv_path_install=${ac_cv_path_install='/usr/bin/install -c'}
ac_cv_prog_CC=${ac_cv_prog_CC='gcc'} ac_cv_prog_CXX=${ac_cv_prog_CXX='c++'}
ac_cv_prog_RANLIB=${ac_cv_prog_RANLIB='ranlib'}
ac_cv_prog_cc_cross=${ac_cv_prog_cc_cross='no'}
ac_cv_prog_cc_g=${ac_cv_prog_cc_g='yes'}
ac_cv_prog_cc_works=${ac_cv_prog_cc_works='yes'}
ac_cv_prog_cxx_cross=${ac_cv_prog_cxx_cross='no'}
ac_cv_prog_cxx_g=${ac_cv_prog_cxx_g='yes'}
ac_cv_prog_cxx_works=${ac_cv_prog_cxx_works='yes'}
ac_cv_prog_gcc=${ac_cv_prog_gcc='yes'}
ac_cv_prog_gxx=${ac_cv_prog_gxx='yes'}
ac_cv_prog_make_make_set=${ac_cv_prog_make_make_set='yes'}
```

LIB_LMPT/config.h.in

```
/* config.h.in. Generated automatically from configure.in by autoheader. */
/* Name of package */
#undef PACKAGE
/* Version number of package */
#undef VERSION
LIB_LMPT/config.h
/* config.h. Generated automatically by configure. */
/* config.h.in. Generated automatically from configure.in by autoheader. */
/* Name of package */
#define PACKAGE "lib_lmpt"
/* Version number of package */
#define VERSION "0.1"
```

LIB_LMPT/config.log

This file contains any messages produced by compilers while running configure, to aid debugging if configure makes a mistake.

```
configure:561:
checking for a BSD compatible install
configure:614: checking whether build environment is sane
configure:671: checking whether make sets ${MAKE}
configure:717: checking for working aclocal
configure:730: checking for working autoconf
configure:743: checking for working automake
configure:756: checking for working autoheader
configure:769: checking for working makeinfo
configure:786: checking for gcc
configure:899: checking whether the C compiler (gcc ) works
configure:915: gcc -o conftest conftest.c 1>&5
configure:941: checking whether the C compiler (gcc )
```

```
is a cross-compiler
configure:946: checking whether we are using GNU C
configure:974: checking whether gcc accepts -g
configure:1010: checking for c++
configure:1042: checking whether the C++ compiler (c++ ) works
configure:1058: c++ -o conftest conftest.C 1>&5
configure:1084: checking whether the C++ compiler (c++ ) is a cross-compiler
configure:1089: checking whether we are using GNU C++
configure:1117: checking whether c++ accepts -g
configure:1151: checking for ranlib
```

LIB_LMPT/config.status

```
#!/bin/sh # Generated automatically by configure.
# Run this file to recreate the current configuration.
```

LIB_LMPT/configure

```
#O arquivo configure é um arquivo de shell grande ( 50k)
#criado pelo autoconf e que quando executado
#faz centenas de testes no sistema do usuário e cria os arquivos Makefile.
# Guess values for system-dependent variables and create Makefiles.

# Generated automatically using autoconf version 2.13
```

LIB_LMPT/lib_lmpt/Makefile.am

```
bin_PROGRAMS = lib_lmpt
lib_lmpt_SOURCES = TTeste.cpp main.cpp
lib_lmpt_LDADD = ./source/Base/libBase.a
SUBDIRS = include source
EXTRA_DIST = main.cpp TTeste.cpp TTeste.h
```

LIB_LMPT/lib_lmpt/include/Makefile.am

```
SUBDIRS = Base
```

LIB_LMPT/lib_lmpt/include/Base/Makefile.am

```
EXTRA_DIST = TMath.h TOperacao.h
```

LIB_LMPT/lib_lmpt/source/Makefile.am

```
SUBDIRS = Base
```

LIB_LMPT/lib_lmpt/source/Base/Makefile.am

```
noinst_LIBRARIES = libBase.a  
libBase_a_SOURCES = TOperacao.cpp TMath.cpp  
EXTRA_DIST = TMath.cpp TOperacao.cpp
```

LIB_LMPT/lib_lmpt/docs/en/Makefile.am

```
EXTRA_DIST = index.html index-1.html index-2.html index-3.html  
index-4.html index-5.html index-6.html
```

Capítulo 13

Introdução ao Controle de Versões Com o CVS

Neste capítulo vamos apresentar o CVS, um sistema para controle das versões de seu programa ou projeto. Vamos ver o que é o cvs, os comandos e a sequência de trabalho.

Este capítulo foi escrito usando as referências [1, 8, 6, 13].

13.1 O que é o CVS¹?

CVS é um sistema de controle de versões (Concurrent Versions System).

- Com CVS você pode gerenciar diferentes versões de um programa (ou projeto).
- Pode atualizar, adicionar e eliminar arquivos e diretórios ao programa.
- Pode criar ramificações de um projeto.
- Múltiplos programadores podem trabalhar ao mesmo tempo no mesmo projeto.
- Informações recentes sobre o CVS você encontra no site (<http://www.cvshome.org/>).

O que é o repositório?

É um diretório com todos os arquivos e subdiretórios do projeto. Adicionalmente, contém arquivos criados pelo programa cvs para o gerenciamento das versões.

O que é uma versão, um tag, um release ?

Todo arquivo tem uma **versão** que é automaticamente definida pelo cvs. Um **tag** é um nome simbólico dado a uma determinada versão do projeto, pode ser usado para delimitar etapas do desenvolvimento de um projeto. Um **release** é uma versão definitiva de todos os arquivos do projeto.

¹O que o CVS não é ? CVS não é um sistema para construção do soft. Não substitui o gerenciamento do soft. Não substitui a necessidade de comunicação entre o grupo de desenvolvimento. Não serve para testar o soft.

O que é um branch (ramo)?

Um branch (ramo) é usado para dividir um projeto. Normalmente existe o ramo mestre e os ramos secundários.

13.2 Comandos do cvs

Veja a seguir o protótipo do programa cvs. Observe que você passa um conjunto de opções para o cvs; depois, o nome do comando a ser executado e um conjunto de argumentos relativos ao comando.

Protocolo:

cvs [cvs-options] command [command-options-and-arguments]

Os principais comandos do cvs são o **cvs checkout** que baixa os arquivos do repositório para seu local de trabalho, o **cvs update** que atualiza os arquivos do local de trabalho, e o **cvs commit**, que devolve ao repositório os arquivos que você modificou.

Lista-se a seguir a saída do comando **cvs - -help-options** que mostra a lista de opções do programa cvs.

Listing 13.1: Saída do comando: cvs - -help-options

```
[andre@mercurio cvs]$ cvs --help-options
CVS global options (specified before the command name) are:
  -H          Displays usage information for command.
  -Q          Cause CVS to be really quiet.
  -q          Cause CVS to be somewhat quiet.
  -r          Make checked-out files read-only.
  -w          Make checked-out files read-write (default).
  -l          Turn history logging off.
  -n          Do not execute anything that will change the disk.
  -t          Show trace of program execution -- try with -n.
  -v          CVS version and copyright.
  -T tmpdir   Use 'tmpdir' for temporary files.
  -e editor   Use 'editor' for editing log information.
  -d CVS_root Overrides $CVSROOT as the root of the CVS tree.
  -f          Do not use the ~/.cvsrc file.
  -z #        Use compression level '#' for net traffic.
  -x          Encrypt all net traffic.
  -a          Authenticate all net traffic.
  -s VAR=VAL  Set CVS user variable.
```

Lista-se a seguir a saída do comando **cvs - -help-commands** o mesmo apresenta a lista de comandos do cvs.

Listing 13.2: Saída do comando: cvs - -help-commands

```
CVS commands are:
  add          Add a new file/directory to the repository
  admin        Administration front end for rcs
  annotate      Show last revision where each line was modified
  checkout     Checkout sources for editing
  commit       Check files into the repository
  diff         Show differences between revisions
  edit         Get ready to edit a watched file
```

editors	See who is editing a watched file
export	Export sources from CVS, similar to checkout
history	Show repository access history
import	Import sources into CVS, using vendor branches
init	Create a CVS repository if it doesn't exist
kserver	Kerberos server mode
log	Print out history information for files
login	Prompt for password for authenticating server
logout	Removes entry in .cvspass for remote repository
pserver	Password server mode
rannotate	Show last revision where each line of module was modified
rdiff	Create 'patch' format diffs between releases
release	Indicate that a Module is no longer in use
remove	Remove an entry from the repository
rlog	Print out history information for a module
rtag	Add a symbolic tag to a module
server	Server mode
status	Display status information on checked out files
tag	Add a symbolic tag to checked out version of files
unedit	Undo an edit command
update	Bring work tree in sync with repository
version	Show current CVS version(s)
watch	Set watches
watchers	See who is watching a file

Para obter um help específico sobre um determinado comando use o comando: **cvsv -H comando**.

Como alguns comandos podem ser repetidos com frequência, os mesmos possuem sinônimos. A listagem a seguir apresenta estes sinônimos.

Listing 13.3: Saída do comando: `cvsv -help-synonyms`

```
[andre@mercurio cvs]$ cvsv --help-synonyms
CVS command synonyms are:
    add          ad new
    admin        adm rcs
    annotate     ann
    checkout     co get
    commit       ci com
    diff         di dif
    export       exp ex
    history      hi his
    import       im imp
    log          lo
    login        logon lgn
    rannotate    rann ra
    rdiff        patch pa
    release      re rel
    remove       rm delete
    rlog         rl
    rtag         rt rfreeze
    status       st stat
    tag          ta freeze
    update       up upd
    version      ve ver
```

13.3 Sequência de trabalho

Apresenta-se nas seções que seguem os comandos e exemplos de uso do cvs.

Primeiro vamos criar o repositório, a seguir vamos importar um projeto antigo (que já existia) para dentro do repositório. Definido o repositório e importado o projeto, podemos iniciar o uso efetivo do cvs. Vamos criar um diretório de trabalho e com o comando checkout copiar os arquivos do repositório para dentro de nosso diretório de trabalho. Vamos aprender a adicionar novos arquivos e diretórios ao projeto. Finalmente, vamos devolver para o repositório os arquivos modificados com o comando commit.

13.3.1 Roteiro para criar um repositório

1. Setar a variável CVSROOT no arquivo profile (ou no arquivo ~/.bash_profile):

```
CVSROOT=/home/REPOSITORY
export CVSROOT
```

Se estiver usando o cshel

```
setenv CVSROOT = /home/REPOSITORY
```

2. A seguir, você deve criar o diretório onde o repositório vai ser armazenado (se necessário, como root):

```
mkdir /home/REPOSITORY
```

- Todos os usuários que vão usar o cvs devem ter acesso a este diretório. A dica é criar um grupo de trabalho com permissão de leitura e escrita ao diretório do repositório.

3. Você pode criar um grupo cvs, adicionar ao grupo cvs os usuários que terão acesso ao repositório e mudar as permissões de acesso ao repositório.

```
chown -R cvs /home/REPOSITORY
chmod g+rwX /home/REPOSITORY
```

- ²A variável CVSUMASK é usada para controlar a forma como os arquivos e diretórios são criados. Consulte um manual de GNU/Linux, Unix, MacOS X para maiores detalhes.

4. O comando **init** inicializa o uso do cvs, adicionando ao diretório do repositório (/home/REPOSITORY) alguns arquivos de controle do programa cvs.

```
cvs init
```

Dê uma olhada no diretório /home/REPOSITORY, observe que foi criado o subdiretório /home/REPOSITORY/CVSROOT. Este subdiretório contém os arquivos de administração do cvs.

- Os arquivos com *,v são read-only.

13.3.2 Para importar os arquivos de seu projeto antigo para dentro do repositório

Você provavelmente já tem um diretório com projetos antigos e com arquivos de programação (*.h, *.cpp). O comando **import** copia o seu diretório para dentro do repositório.

Protótipo:

```
cd path_completa_projeto_antigo
cvs import -m "mensagem" path_proj_no_repositorio nome_release nome_tag
```

-m "msg" É uma mensagem curta contendo informação sobre o projeto.

path_proj_no_repositorio É a path para o diretório do projeto no repositório.

nome_release É o nome do release inicial.

nome_tag Informa o tag inicial do projeto (normalmente = start).

Vamos adicionar ao repositório o projeto exemplo-biblioteca-gnu localizado, em minha máquina, no diretório:

~/ApostilaProgramacao/Exemplos/Cap-GNU/biblioteca.

```
cd ~/ApostilaProgramacao/Exemplos/Cap-GNU/biblioteca
cvs import -m "Exemplo de biblioteca usando ferramentas gnu"
                exemplo-biblioteca-gnu R1 start
```

A saída gerada pelo comando import é apresentada na listagem a seguir. Observe que a letra N indica um arquivo novo, a letra I um arquivo ignorado (arquivos *.bak *.~ são ignorados pelo cvs). A biblioteca recebe um L de library.

Listing 13.4: Saída do comando: cvs import

```
[andre@mercurio biblioteca]$ cvs import -m "Exemplo de biblioteca usando
ferramentas gnu" exemplo-biblioteca-gnu R1 start
N exemplo-biblioteca-gnu/e87-Polimorfismo.cpp
N exemplo-biblioteca-gnu/e87-Programa.cpp
N exemplo-biblioteca-gnu/e87-TCirculo.cpp
I exemplo-biblioteca-gnu/doxygem.config.bak
N exemplo-biblioteca-gnu/makefile
N exemplo-biblioteca-gnu/e87-TCirculo.h
N exemplo-biblioteca-gnu/doxygem.config
N exemplo-biblioteca-gnu/uso-makefile
N exemplo-biblioteca-gnu/e87-PolimorfismoStatic.cpp
N exemplo-biblioteca-gnu/e87-TElipse.cpp
N exemplo-biblioteca-gnu/e87-TElipse.h
N exemplo-biblioteca-gnu/e87-PolimorfismoDinamic.cpp
N exemplo-biblioteca-gnu/Makefile
N exemplo-biblioteca-gnu/e87-TPonto.cpp
N exemplo-biblioteca-gnu/e87-TPonto.h
N exemplo-biblioteca-gnu/e87-Polimorfismo
I exemplo-biblioteca-gnu/e87-Polimorfismo.cpp~
N exemplo-biblioteca-gnu/makefile-libtool
cvs import: Importing /home/REPOSITORY/exemplo-biblioteca-gnu/.libs
N exemplo-biblioteca-gnu/.libs/libTPonto.al
```

```
L exemplo-biblioteca-gnu/.libs/libTPonto.la
```

```
No conflicts created by this import
```

Você pode executar o comando **ls /home/REPOSITORY** ou **tree /home/REPOSITORY** para ver como os arquivos foram importados para dentro do repositório.

Listing 13.5: Como fica o repositório após a importação

```
/home/REPOSITORY/
|-- CVSROOT
|   |-- modules
|   |-- notify
|   |-- .....
|   '-- verifymsg,v
'-- exemplo-biblioteca-gnu
    |-- Makefile,v
    |-- doxygem.config,v
    |-- doxygem.configold,v
    |-- e87-Polimorfismo,v
    |-- e87-Polimorfismo.cpp,v
    |-- e87-PolimorfismoDinamic.cpp,v
    |-- e87-PolimorfismoStatic.cpp,v
    |-- e87-Programa.cpp,v
    |-- e87-TCirculo.cpp,v
    |-- e87-TCirculo.h,v
    |-- e87-TElipse.cpp,v
    |-- e87-TElipse.h,v
    |-- e87-TPonto.cpp,v
    |-- e87-TPonto.h,v
    |-- makefile,v
    |-- makefile-funciona,v
    |-- makefile-libtool,v
    |-- makefile-ok,v
    '-- uso-makefile,v
```

Dica: Depois de importar seus projetos para dentro do repositório, faça um backup dos projetos (tar -cvzf NomeProjeto.tar.gz NomeProjeto) e remova os arquivos do projeto (rm -fr NomeProjeto). Desta forma você elimina a possibilidade de trabalhar acidentalmente nos arquivos de seu projeto em vez de trabalhar com os arquivos do repositório.

13.3.3 Para baixar o projeto

O nosso repositório já foi criado, já definimos um grupo de trabalho e já copiamos para dentro do repositório um projeto. Agora vamos iniciar o uso efetivo do cvs.

Para copiar os arquivos de dentro do repositório para o diretório onde você deseja trabalhar, usa-se o comando **checkout**. Veja na listagem a seguir o protótipo e os parâmetros do comando **checkout**.

Listing 13.6: Saída do comando: cvs -H checkout

```
[andre@mercurio cvs]$ cvs -H checkout
Usage:
  cvs checkout [-ANPRcflnps] [-r rev] [-D date] [-d dir]
               [-j rev1] [-j rev2] [-k kopt] modules...
```

```
-A      Reset any sticky tags/date/kopts.
-N      Don't shorten module paths if -d specified.
-P      Prune empty directories.
-R      Process directories recursively.
-c      "cat" the module database.
-f      Force a head revision match if tag/date not found.
-l      Local directory only, not recursive
-n      Do not run module program (if any).
-p      Check out files to standard output (avoids stickiness).
-s      Like -c, but include module status.
-r rev  Check out revision or tag. (implies -P) (is sticky)
-D date Check out revisions as of date. (implies -P) (is sticky)
-d dir  Check out into dir instead of module name.
-k kopt Use RCS kopt -k option on checkout. (is sticky)
-j rev  Merge in changes made between current revision and rev.
(Specify the --help global option for a list of other help options)
```

Vá para o diretório onde deseja trabalhar e crie uma cópia de trabalho com **checkout**.

```
Exemplo
mkdir /tmp/workdir
cd /tmp/workdir
cvs checkout exemplo-biblioteca-gnu
cd exemplo-biblioteca-gnu
ls -la
```

Observe que todos os arquivos do projeto foram copiados para o diretório /tmp/workdir/exemplo-biblioteca-gnu. Também foi criado o diretório cvs. Este diretório é mantido pelo programa cvs.

13.3.4 Para criar módulos

Bem, com o comando checkout, fizemos uma cópia de trabalho do projeto exemplo-biblioteca-gnu. Mas o nome exemplo-biblioteca-gnu é muito extenso e seria melhor um nome abreviado. Um módulo é exatamente isto, um nome abreviado para uma path grande no diretório do repositório. Veja a seguir como criar um módulo.

1. Baixa o arquivo modules, localizado em /home/REPOSITORY/CVSRROOT/modules

```
cvs checkout CVSRROOT/modules
```

2. Edita o arquivo modules

```
emacs CVSRROOT/modules
```

3. Inclua a linha abaixo (nome_módulo path)

```
lib-gnu exemplo-biblioteca-gnu
```

4. Salva o arquivo e envia para o repositório com o comando

```
cvs commit -m
    "adicionado o módulo exemplo-biblioteca-gnu ->lib-gnu"
```

O comando **commit** é usado para devolver para o repositório todos os arquivos novos ou modificados. Veja na listagem a seguir o protótipo do comando commit.

Listing 13.7: Saída do comando: cvs -H commit

```
[andre@mercurio cvs]$ cvs -H commit
Usage: cvs commit [-nRlf] [-m msg | -F logfile] [-r rev] files...
  -n          Do not run the module program (if any).
  -R          Process directories recursively.
  -l          Local directory only (not recursive).
  -f          Force the file to be committed; disables recursion.
  -F logfile  Read the log message from file.
  -m msg      Log message.
  -r rev      Commit to this branch or trunk revision.
(Specify the --help global option for a list of other help options)
```

Veja na listagem a seguir a saída do comando commit executada no diretório de trabalho após a modificação do arquivo CVSROOT/modules.

Listing 13.8: Saída do comando cvs commit após adição de um módulo

```
[andre@mercurio workdir]$ cvs commit -m "adicionado o módulo exemplo-biblioteca-
gnu -> lib-gnu"
cvs commit: Examining CVSROOT
cvs commit: Examining exemplo-biblioteca-gnu
cvs commit: Examining exemplo-biblioteca-gnu/.libs
Checking in CVSROOT/modules;
/home/REPOSITORY/CVSROOT/modules,v <-- modules
new revision: 1.2; previous revision: 1.1
done
cvs commit: Rebuilding administrative file database
```

Agora você pode executar o comando checkout de forma abreviada, usando o nome do módulo.

```
mkdir /tmp/workdir2
cd /tmp/workdir2
cvs checkout lib-gnu
```

Para que o comando ficasse ainda mais curto, poderia-se ter utilizado a forma abreviada de checkout.

```
cvs co lib-gnu
```

13.3.5 Para adicionar/remover arquivos e diretórios

O comando **add** agenda a adição de arquivos e diretórios que só serão copiados para o repositório com o comando **commit**. Da mesma forma, o comando **remove** agenda a remoção de arquivos e diretórios que só serão removidos do repositório com o comando **commit**.

Veja a seguir o protótipo destes comandos. Observe que para os comandos funcionarem, você deve estar no diretório de trabalho (/tmp/workdir).

Para adicionar um arquivo

Vamos criar um arquivo `leiametext`, o mesmo contém alguma informação sobre o projeto. Vamos criá-lo com o editor `emacs` (use o que lhe convier).

```
emacs leiametext
...inclui observações no arquivo leiametext...
```

Agora vamos agendar a adição do arquivo com o comando `add`. A saída do comando é apresentada em *itálico*.

```
cvsc add -m "adicionado arquivo leiametext" leiametext
cvsc add: scheduling file 'leiametext' for addition
cvsc add: use 'cvsc commit' to add this file permanently
```

Depois de modificar outros arquivos, podemos efetivamente adicionar o arquivo `leiametext` no repositório usando o comando `commit`. Observe, em *itálico*, a saída gerada pelo comando `commit`.

```
cvsc commit
cvsc commit: Examining .
cvsc commit: Examining .libs
cvsc commit: Examining novoDir
RCS file:/home/REPOSITORY/exemplo-biblioteca-gnu/leiametext,v done
Checking in leiametext;
/home/REPOSITORY/exemplo-biblioteca-gnu/leiametext,v <--
leiametext initial revision: 1.1
done
```

Alguns comandos do programa `cvsc` podem abrir um editor de texto para que você inclua alguma mensagem relativa a operação que foi realizada. No exemplo acima, depois do **cvsc commit**, o `cvsc` abriu o editor `emacs`. Na sua máquina provavelmente irá abrir o `vi`. Você pode alterar o editor a ser aberto pelo `cvsc`, setando no arquivo `~/.bash_profile` a variável de ambiente `CVSEEDITOR` (Em minha máquina: `export CVSEEDITOR=emacs`).

Para adicionar vários arquivos:

O procedimento é o mesmo, primeiro agenda a adição com `add` e depois adiciona efetivamente com `commit`.

```
cvsc add -m "adicionados diversos arquivos" *
cvsc commit
```

Para adicionar um diretório:

A sequência envolve a criação do diretório (`mkdir novoDir`), o agendamento da adição (`cvsc add novoDir`), e a efetiva adição do diretório com `commit`.

```
mkdir novoDir
cvsc add novoDir
cvsc commit -m "adicionado novo diretório" novoDir
```

Para adicionar toda uma estrutura de diretórios num projeto existente:

É o mesmo procedimento utilizado para importar todo um projeto. A única diferença é que a path de importação no repositório vai estar relativa a um projeto já existente. Veja o exemplo:

```
cd novoDir
cvs import -m "msg" path_proj_no_repositorio/novodir
                           nome_release nome_tag.
```

Para remover um arquivo:

Você deve remover o arquivo localmente, agendar a remoção e então efetivar a remoção com commit.

```
rm leiametext
cvs remove leiametext
cvs commit leiametext
```

O comando a seguir remove o arquivo localmente e no cvs ao mesmo tempo.

```
cvs remove -f leiametext
```

Para remover vários arquivos:

Você deve remover os arquivos, agendar a remoção e então remover efetivamente com commit.

```
rm -f *
cvs remove
cvs commit -m "removidos diversos arquivos"
```

Dica: Se você fizer alterações locais em um arquivo e depois remover o arquivo, não poderá recuperá-las. Para que possa recuperar as alterações, deve criar uma versão do arquivo usando o comando commit.

Para remover diretórios:

Vá para dentro do diretório que quer deletar, e delete todos os arquivos e o diretório usando:

```
cd nomeDir
cvs remove -f *
cvs commit
//A seguir delete o diretório:
cd ..
cvs remove nomeDir/
cvs commit
```

Para renomear arquivos:

Vá para dentro do diretório onde está o arquivo a ser renomeado e execute os passos:

```
cd diretorio
mv nome_antigo nome_novo
cvs remove nome_antigo
cvs add nome_novo
cvs commit -m "Renomeado nome_antigo para nome_novo"
```

13.3.6 Para atualizar os arquivos locais

Como o cvs permite o trabalho em grupo. Um segundo usuário pode ter copiado e alterado os arquivos do projeto no repositório.

Um segundo usuário realizou as tarefas a seguir²:

```
mkdir /tmp/workdir3
cd /tmp/workdir3
cvs checkout lib-gnu
cd lib-gnu
emacs arquivo-usuario2.txt
cvs add arquivo-usuario2.txt
cvs commit -m "arquivo adicionado pelo usuario2"
```

Se outros usuários do projeto modificaram os arquivos do repositório, então os arquivos com os quais você está trabalhando podem estar desatualizados. Isto é, se um outro usuário modificou algum arquivo do repositório, você precisa atualizar os arquivos em seu diretório de trabalho.

Bastaria realizar um comando **cvs commit** devolvendo para o repositório todos os arquivos que você modificou, e um comando **cvs checkout**, que copiaria todos os arquivos do repositório, atualizados, para seu diretório de trabalho. Mas este procedimento pode ser lento. Seria mais rápido se o cvs copia-se para seu diretório de trabalho apenas os arquivos novos e modificados. É exatamente isto que o comando **update** faz. O protótipo do comando update é listado a seguir.

Listing 13.9: Saída do comando: cvs -H update

```
[andre@mercurio cvs]$ cvs -H update
Usage: cvs update [-APCdfIRp] [-k kopt] [-r rev] [-D date] [-j rev]
        [-I ign] [-W spec] [files...]
  -A      Reset any sticky tags/date/kopts.
  -P      Prune empty directories.
  -C      Overwrite locally modified files with clean repository copies.
  -d      Build directories, like checkout does.
  -f      Force a head revision match if tag/date not found.
  -l      Local directory only, no recursion.
  -R      Process directories recursively.
  -p      Send updates to standard output (avoids stickiness).
  -k kopt Use RCS kopt -k option on checkout. (is sticky)
  -r rev  Update using specified revision/tag (is sticky).
  -D date Set date to update from (is sticky).
  -j rev  Merge in changes made between current revision and rev.
  -I ign  More files to ignore (! to reset).
  -W spec Wrappers specification line.
```

Veja no exemplo como deixar seu diretório de trabalho com os arquivos atualizados.

```
cd /tmp/workdir
cvs update
```

²Observe que o nome do diretório obtido pelo usuário 1 é exemplo-biblioteca-gnu e do usuário 2 lib-gnu. Isto é, se você usa `cvs checkout path_proj_no_repositorio` o cvs cria o diretório `path_proj_no_repositorio`. Se você usa `cvs checkout nome_modulo`, o cvs cria o diretório `nome_modulo`.

```

cv$ update: Updating . U arquivo-usuario2.txt
cv$ update: Updating .libs
cv$ update: Updating novoDir

```

Observe que o arquivo “arquivo-usuario2.txt” criado pelo usuário 2 foi adicionado a sua cópia de trabalho.

13.4 Versões, tag's e releases

Descrevemos no início deste capítulo o que é um release e um tag. Apresenta-se a seguir como criar e usar releases e tags.

13.4.1 Entendendo as versões

Todos os arquivos do projeto que foram importados ou adicionados ao repositório tem uma versão. A versão é definida automaticamente pelo programa cvs e se aplica aos arquivos individualmente, isto é, cada arquivo tem sua versão.

De uma maneira geral a versão do arquivo é redefinida a cada alteração do arquivo que foi comutada com o repositório. Assim, se o arquivo leiname.txt, que tem a versão 1.1, foi alterado. Quando o mesmo for devolvido ao repositório com o comando **cv\$ commit**, o mesmo passa a ter a versão 1.2. Veja Figura ??.

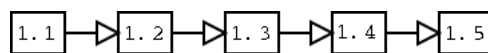


Figura 13.1: Versões de um arquivo.

No exemplo a seguir vai para o diretório de trabalho e modifica o arquivo leiname.txt. Depois realiza um commit. Observe a alteração na versão.

```

cd /tmp/workdir/exemplo-biblioteca-gnu
emacs leiname.txt
...faça alterações no arquivo leiname.txt...e depois salve o arquivo.
cv$ commit

cv$ commit: Examining .
cv$ commit: Examining .libs
cv$ commit: Examining novoDir
Checking in leiname.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/leiname.txt,v <-- leiname.txt
new revision: 1.2; previous revision: 1.1 done

```

13.4.2 Para criar tag's

Como dito acima, cada arquivo do repositório vai ter uma versão. Entretanto, você pode realizar diversas modificações no arquivo leiname.txt (1.1 -> 1.2 -> 1.3 -> 1.4 -> 1.5), algumas modificações no arquivo makefile (1.1 -> 1.2 -> 1.3) e nenhuma modificação no arquivo NomePrograma.cpp (1.1). Ou

seja, cada arquivo tem um número de versão diferente. Seria interessante se você pudesse se referir a todos os arquivos do projeto em uma determinada data com um mesmo nome simbólico. Um **tag** é exatamente isto, um nome simbólico usado para obter os arquivos do projeto em determinada data.

Veja na Figura ?? como é criado um novo tag. Observe que a versão de cada arquivo não é alterada.

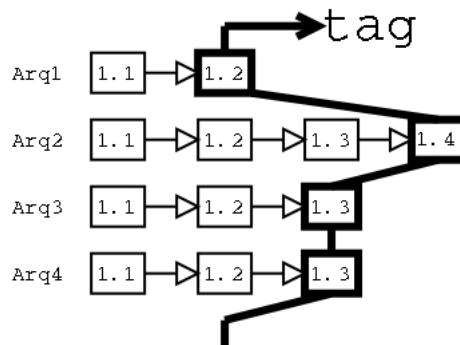


Figura 13.2: Criando um tag.

Assim, em determinado dia eu quero criar um tag simbólico, um nome que vou utilizar para todos os arquivos do projeto naquela data.

Protótipo para criar um tag para um único arquivo:

```
cd /tmp/workdir
cvs tag nome_release_simbólico nome_arquivo
```

Protótipo para criar um tag para todos os arquivos do projeto:

```
cd /tmp/workdir
cvs tag nome_release_simbólico
```

Veja na listagem a seguir a saída do comando, `cvs tag tag1 *` executada em nosso diretório de trabalho.

Listing 13.10: Saída do comando: `cvs -tag nome`

```
[andre@mercurio exemplo-biblioteca-gnu]$ cvs tag tag1 *
cvs tag: warning: directory CVS specified in argument
cvs tag: but CVS uses CVS for its own purposes; skipping CVS directory
T arquivo-usuario2.txt
T doxygem.config
T e87-Polimorfismo
T e87-Polimorfismo.cpp
T e87-PolimorfismoDinamic.cpp
T e87-PolimorfismoStatic.cpp
T e87-Programa.cpp
T e87-TCirculo.cpp
T e87-TCirculo.h
T e87-TElipse.cpp
T e87-TElipse.h
T e87-TPonto.cpp
T e87-TPonto.h
T leiname.txt
```

```
T makefile
T Makefile
T makefile-libtool
T uso-makefile
cvs tag: Tagging novoDir
```

Para recuperar a versão completa do projeto usando o tag que acabamos de criar:

```
cd /tmp/workdir/exemplo-biblioteca-gnu
cvs checkout -r tag1 lib-gnu
```

Observe que para baixar o módulo lib-gnu usamos **cvs checkout lib-gnu**, e para baixar o tag1 do módulo lib-gnu, usamos, **cvs checkout -r tag1 lib-gnu**. Ou seja, apenas adicionamos após o comando checkout, o parâmetro -r e o nome do tag.

13.4.3 Para criar release's

Geralmente utilizamos um tag para criar uma versão do projeto que esteja funcionando, ou que compreenda a finalização de um determinado conjunto de tarefas que estavam pendentes. Assim, com o nome do tag você pode recuperar o projeto naquela data usando um nome abreviado.

Entretanto, depois de finalizado o programa ou uma versão funcional, você pode criar um release do programa. A diferença entre o tag e o release, é que o tag não modifica a versão dos arquivos do projeto. O release modifica a versão de todos os arquivos, dando a todos os arquivos um mesmo número de versão. Veja na Figura ?? como fica um novo release.

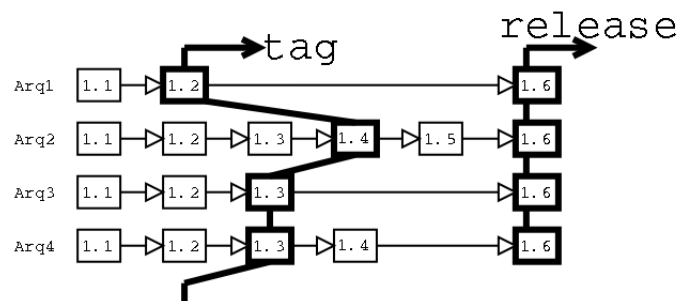


Figura 13.3: Criando um release.

- Um release é geralmente um pacote funcional, se aplica a todos os arquivos do projeto.
- Depois de definido o release o mesmo não pode ser modificado.
- Você deve criar um release sempre que tiver finalizado uma parte importante de seu programa.

Veja a seguir o protótipo para criar um release.

Protótipo:

```
cvs commit -r número_release
```

```
cd /tmp/workdir
cvs commit -r 2
```

Além de criar o release, abre o `vi`³, para edição de um arquivo de log. Inclua algum comentário a respeito do release que foi criado.

Veja na listagem a seguir a saída do comando **cvs commit -r 2**. Observe que todos os arquivos passam a ter a mesma versão.

Listing 13.11: Saída do comando: `cvs commit -r 2`

```
[root@mercurio lib-gnu]# cvs commit -r 2
cvs commit: Examining .
cvs commit: Examining .libs
cvs commit: Examining novoDir
Checking in Makefile;
/home/REPOSITORY/exemplo-biblioteca-gnu/Makefile,v <-- Makefile
new revision: 2.1; previous revision: 1.1
done
Checking in arquivo-usuario2.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/arquivo-usuario2.txt,v <-- arquivo-
    usuario2.txt
new revision: 2.1; previous revision: 1.1
done
....
....
Checking in leiamе.txt;
/home/REPOSITORY/exemplo-biblioteca-gnu/leiamе.txt,v <-- leiamе.txt
new revision: 2.1; previous revision: 1.3
done
```

Protótipo para criar um release e já deletar a cópia do diretório local:

```
cvs release -d diretório_de_trabalho
```

13.4.4 Recuperando módulos e arquivos

O cvs permite que tanto os códigos novos como os antigos possam ser recuperados. De uma maneira geral basta passar o nome do arquivo e sua versão (tag, release, módulo).

Protótipo para recuperar um release:

```
#Pode-se baixar um release antigo, passando o nome do release.
cvs checkout -r nome_release path_projeto_no_cvs
#ou o nome do módulo
cvs checkout -r nome_release nome_modulo
```

Protótipo para recuperar um arquivo de uma versão antiga:

```
cvs update -p -r nome_release nome_arquivo > nome_arquivo
```

-p Envia atualizações para saída padrão (a tela).

-r nome_release Indica a seguir o nome do release.

³ou o editor setado com CVSEDITOR. No vi digite `esc :q` para sair, `esc :q!` para sair sem salvar alterações.

nome_arquivo O nome do arquivo a ser baixado

> *nome_arquivo* Redireciona da tela para o arquivo *nome_arquivo*.

No exemplo a seguir, recupera o arquivo *leiametext* do *tag1*.

```
cvs update -p -r tag1 leiametext > leiametext-tag1.txt
```

```
=====
Checking out leiametext RCS:
/home/REPOSITORY/exemplo-biblioteca-gnu/leiametext,v VERS: 1.2
*****
```

13.5 Para verificar diferenças entre arquivos

O programa *cvs* tem suporte interno ao programa *diff* (apresentado no Capítulo ??), permitindo comparar os arquivos que estão sendo usados localmente com os do repositório.

Protótipo:

#Compara arq local e arq do repositório

cvs diff arq

#Verifica diferenças de todos os arquivos

cvs diff

O usuário 2, modificou o arquivo *leiametext* depois de criado o *release 2*. Veja na listagem a seguir a saída do comando *cvs diff*, executado pelo usuário 1.

Listing 13.12: Saída do comando: *cvs-diff*

```
[andre@mercurio exemplo-biblioteca-gnu]$ cvs diff
cvs diff: Diffing .
Index: leiametext
=====
RCS file: /home/REPOSITORY/exemplo-biblioteca-gnu/leiametext,v
retrieving revision 2.2
diff -r2.2 leiametext
7,11d6
< Alteração realizada depois de criado o tag1.
<
<
< Modificações realizadas depois do release.
< Pelo usuário 2.
cvs diff: Diffing .libs
cvs diff: Diffing novoDir
```

13.6 Verificando o estado do repositório

O *cvs* tem um conjunto de comandos que você pode usar para verificar o estado dos arquivos armazenados no repositório.

13.6.1 Histórico das alterações

Você pode obter uma lista com o histórico das alterações realizadas.

Mostra: data, hora, usuário, path usada (ou módulo, ou ramo), diretório de trabalho:

Protótipo:

cvs history

13.6.2 Mensagens de log

Você pode obter uma lista dos log's do arquivo.

Mostra: path no repositório, versão, nomes simbólicos, revisões e anotações realizadas.

Protótipo:

cvs log arquivo

Veja a seguir a saída do comando **cvs -log leiname.txt**. Observe as diferentes revisões e anotações, o nome do autor. Observe nos nomes simbólicos o tag1 e a referencia a versão 1.2, isto significa que quando o usuário solicitar o tag1, corresponde a versão 1.2 do arquivo leiname.txt.

Listing 13.13: Saída do comando: cvs -log leiname.txt

```
RCS file: /home/REPOSITORY/exemplo-biblioteca-gnu/leiname.txt,v
Working file: leiname.txt
head: 2.2
branch:
locks: strict
access list:
symbolic names:
    tag1: 1.2
keyword substitution: kv
total revisions: 5;      selected revisions: 5
description:
adicionado arquivo leiname.txt
-----
revision 2.2
date: 2002/08/12 23:28:55;  author: andre;  state: Exp;  lines: +4 -0
Modificações realizadas no leiname.txt depois de criado o release.
-----
revision 2.1
date: 2002/08/12 23:12:05;  author: andre;  state: Exp;  lines: +0 -0
Criado o release 2.
-----
revision 1.3
date: 2002/08/12 23:10:32;  author: andre;  state: Exp;  lines: +1 -0
Alterações no leiname.txt depois de criado o tag1.
-----
revision 1.2
date: 2002/08/12 22:45:56;  author: andre;  state: Exp;  lines: +5 -0
Modificações no arquivo leiname.txt
-----
revision 1.1
date: 2002/08/12 21:33:43;  author: andre;  state: Exp;
Efetivamente adicionado o arquivo leiname.txt
=====
```

13.6.3 Anotações

Você pode obter uma lista das anotações realizadas.
Mostra: versão, nome usuário, data, mensagem.

Protótipo:

cvs annotate

13.6.4 Verificando o status dos arquivos

O comando status mostra uma série de informações a respeito do arquivo. O mesmo pode ser utilizado para verificar quais arquivos precisam ser atualizados. Veja a seguir o protótipo.

Protótipo:

cvs status

-v *Mostra ainda os tag's.*
-R *Processamento recursivo.*
-l *Somente este diretório.*

Informações listadas pelo comando status:

Up-to-date O arquivo não foi alterado.

Locally modified O arquivo foi modificado localmente.

Locally added O arquivo foi adicionado localmente.

Locally removed O arquivo foi removido localmente.

Needs checkout O arquivo foi alterado por terceiro e precisa ser atualizado (Com um update baixa o arquivo mesclando-o com o local. Com um commit atualiza no servidor).

File had conflicts on merge O arquivo apresenta conflitos após a mistura.

Veja na listagem a seguir a saída do comando status. Observe que o arquivo foi localmente modificado.

Listing 13.14: Saída do comando: `cvs -status leiname.txt`

```
[andre@mercurio exemplo-biblioteca-gnu]$ cvs status leiname.txt
=====
File: leiname.txt                      Status: Locally Modified

Working revision:        2.2            Result of merge
Repository revision: 2.2            /home/REPOSITORY/exemplo-biblioteca-gnu/leiname.
txt,v
Sticky Tag:              (none)
Sticky Date:             (none)
Sticky Options:          (none)
```

13.7 Ramos e Misturas (Branching and Merging)

O programa cvs permite que você crie um ramo principal para seu projeto e ramos derivados. Posteriormente você pode misturar os diferentes ramos.

Veja na Figura ?? a disposição de um novo ramo.

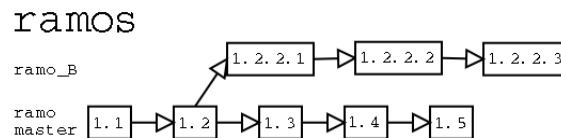


Figura 13.4: Como ficam os ramos.

Depois de finalizado um release de um programa, é bastante usual a criação de três ramos.

Digamos que você esteja trabalhando no projeto gnome, e que o release 1.0 já foi suficientemente testado, podendo ser publicado. Então, você cria o release 1.0.

Release gnome-1.0.

Observe que é a versão final do gnome 1.0.

Agora você pode criar um ramo de patch, o mesmo vai conter os arquivos da versão 1.0, mas com correções de bugs que tenham sido localizados. Assim, se foi identificado algum bug na versão 1.0, você faz as alterações no ramo gnome 1.0-patch, deixando o release 1.0 inalterado.

Ramo: gnome-1.0-patch

Você pode criar um ramo novo, onde ficarão os arquivos da nova versão do gnome.

Ramo: gnome-1.1

Ou seja, vamos ter três ramos. O release 1.0 que não será mais alterado. O patch que vai ter as correções de bugs da versão 1.0 e o 1.1 que terá os arquivos da nova geração do gnome.

13.7.1 Trabalhando com ramos

Para criar um ramo a partir da cópia de trabalho local (-b de branch):

```
cvs tag -b nome_do_ramo
```

Para criar um ramo a partir de um release existente, sem uma cópia de trabalho local:

```
cvs rtag -b -r nome_do_release nome_do_ramo path_no_repositorio
```

Baixando um ramo:

```
cvs checkout -r nome_do_ramo path_no_repositorio
```

Atualização dos arquivos locais de um dado ramo:

```
cvs update -r nome_do_amo path_no_repositorio
```

ou

```
cvs update -r nome_do_amo nome_modulo
```

Para saber com qual ramo você está trabalhando, verifique o nome do ramo em “Existing tags”.

```
cvs status -v nome_arquivo
```

13.7.2 Mesclando 2 versões de um arquivo

Com a opção `-j`, você pode verificar as diferenças entre 2 versões de um arquivo. Veja o protótipo e um exemplo a seguir.

Protótipo:

```
cvs update -j versãoNova -j versãoVelha nomeArquivo
```

```
cvs update -j 2 -j tag1 leiname.txt
```

```
U leiname.txt
```

```
RCS file: /home/REPOSITORY/exemplo-biblioteca-gnu/leiname.txt,v
```

```
retrieving revision 2.2
```

```
retrieving revision 1.2
```

```
Merging differences between 2.2 and 1.2 into leiname.txt
```

Observe a mensagem apresentada. O cvs recupera a versão 2.2 (relativa ao release `-j 2`) e a versão 1.2 (relativa ao `tag1`) e mistura as duas no arquivo `leiname.txt`.

13.7.3 Mesclando o ramo de trabalho com o ramo principal

Digamos que você está trabalhando no ramo principal. Que um segundo usuário criou o ramo `_B` e fez alterações no ramo `_B`. Agora você quer incluir as alterações do ramo `_B` no ramo principal.

1. Baixa o módulo de trabalho

```
cvs checkout nome_modulo
```

2. Baixa o upgrade do ramo `_B`. Ou seja, atualiza os arquivos locais mesclando os mesmos com os do ramo `_B`.

```
cvs update -j ramo_B
```

3. Resolve os possíveis conflitos. Alguns arquivos que tenham sido modificados por outros usuários podem ter conflitos de código, você precisa resolver estes conflitos.

```
Correção de possíveis conflitos de código...
```

4. Copia os arquivos de volta para o repositório, atualizando o repositório.

```
cvs commit -m "Ramo mestre mesclado com ramo_B"
```

5. Para deletar o diretório local de trabalho (use com cuidado).

```
rm -f -r path_local/
```

13.8 Configuração do cvs no sistema cliente-servidor³

Neste tipo de configuração o projeto principal fica na máquina servidora (ou seja o repositório fica no servidor). O usuário baixa o programa para sua máquina local usando checkout, faz modificações e depois copia as modificações para o repositório usando o comando commit.

- O servidor para uso do cvs pode ser um micro pouco potente (133MHz, 32Mb), com HD suficiente (4 vezes o tamanho do projeto).
- O acesso ao repositório é dado por:
:tipo_de_acesso:path_do_projeto

onde tipo_de_acesso:

:local: Você esta na máquina servidora: Se estiver trabalhando na mesma máquina do repositório, você faz um acesso local ao projeto e pode acessar os arquivos do projeto diretamente com **cvs checkout path_no_repositorio**.

:servidor: Você esta na máquina cliente: Se estiver remoto, deve-se incluir o nome do servidor
: **servidor: user@hostname:/path/to/repository**

Ex:

export CVSROOT=:pserver: usuario1@nome_servidor:/path_repositorio
cvs checkout path_no_repositorio.

- Consulte o manual do cvs para ver como configurar o servidor.

Exemplo:

Por default, a conexão do cvs usa o protocolo RSH. Assim, se andre esta na máquina mercurio.lmpt.ufsc.br e o servidor é enterprise.lmpt.ufsc.br no arquivo '.rhosts' deve ter a linha:

```
mercurio.lmpt.ufsc.br andre
```

Para testar:

```
rsh -l bach enterprise.lmpt.ufsc.br 'echo $PATH'
```

Deve-se setar na máquina cliente o endereço do programa cvs no servidor com a variável de ambiente CVS_SERVER.

13.8.1 Variáveis de ambiente

Variáveis de ambiente do cvs definidas no arquivo profile:

\$CVSROOT Diretório de trabalho do cvs.

\$CVS_SERVER Endereço do programa cvs na máquina servidora.

\$CVSEEDITOR Editor default do cvs.

\$CVSUMASK Define o formato dos arquivos novos a serem criados.

13.9 Como baixar programas de terceiros usando o cvs

Veja a seguir as instruções postadas no site <http://www.devel.lyx.org/cvs.php3> e que são utilizadas para baixar o programa LyX usando o cvs.

Anonymous CVS login

In order to make anonymous CVS access easier, you should set your CVSROOT environment variable to

```
:pserver:anoncvs@anoncvs.lyx.org:/usr/local/lyx/cvsroot
```

(Alternatively, you can use the -d option to the cvs commands.).

Now just do:

```
cvs login
```

The password is lyx. Finally,

```
cvs checkout lyx-devel
```

This will make a directory lyx-devel and download lots of files into that directory. Of course you can say cvs checkout lyx-1_0_x instead, depending on which module you'd like to download.

13.10 Frontends

Existem front-ends para o programa cvs. Para o GNU/Linux, de uma olhada no cervisia, encontrado no site (<http://cervisia.sourceforge.net/>) e ilustrado na Figura ???. Um frontend para o Windows é o wincvs, ilustrado na Figura ???.

13.11 Sentenças para o cvs

- Quando você passa como parâmetro de algum comando do cvs um diretório. Todos os arquivos do diretório e subdiretórios sofrem o efeito do comando.
- Uma path, um módulo, um ramo são equivalentes. Um tag, um release, uma versão são equivalentes. Ou seja, se o programa cvs espera uma path_do_repositório você também pode passar o nome de um módulo ou de um ramo. Se o cvs espera um nome de versão, você pode passar o nome do tag, ou o nome do release.
- Monte um grupo de trabalho:
Para trabalhar em grupo em um projeto, você deve-se definir um grupo no GNU/Linux, Unix, MacOS X. O grupo de trabalho terá acesso aos arquivos do repositório num sistema cliente-servidor.
- Pode-se definir diferentes formas de acesso aos arquivos, autenticações e sistemas de segurança. Dê uma olhada no manual de configuração do cvs.
- Links para cvs: <http://www.cvshome.org/>, a casa do cvs. Outros frontends podem ser encontrados em <http://www.cvsgui.org/download.html>, e <http://www.lincvs.org>.

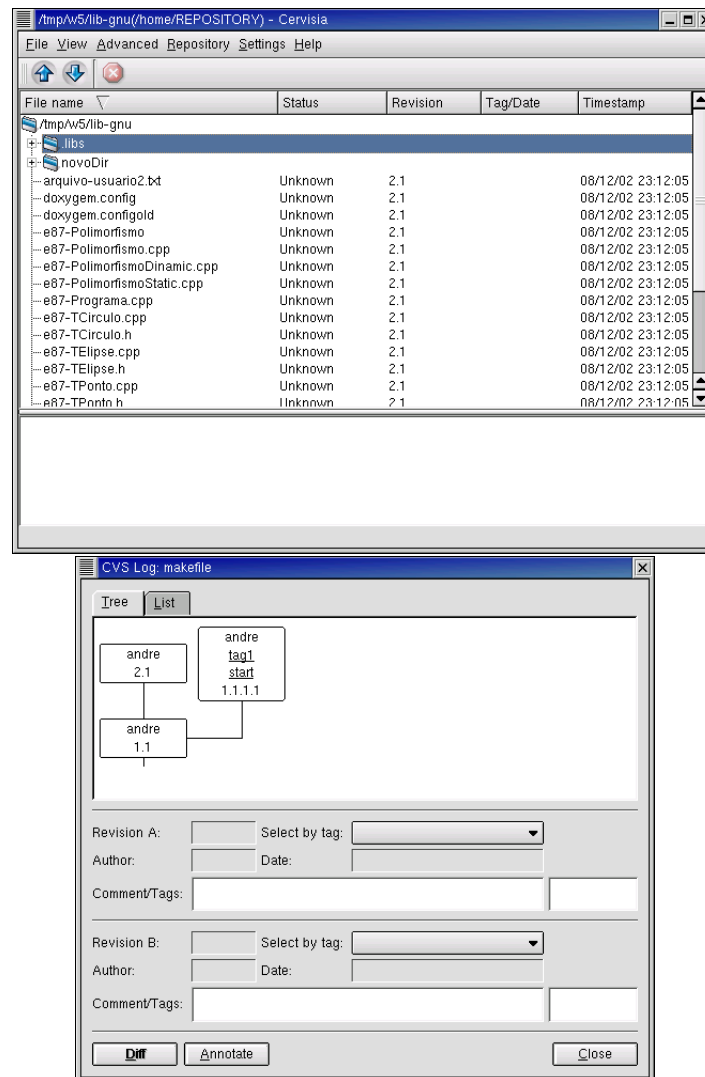


Figura 13.5: Um frontend para o cvs no GNU/Linux, Unix (o cervisia).

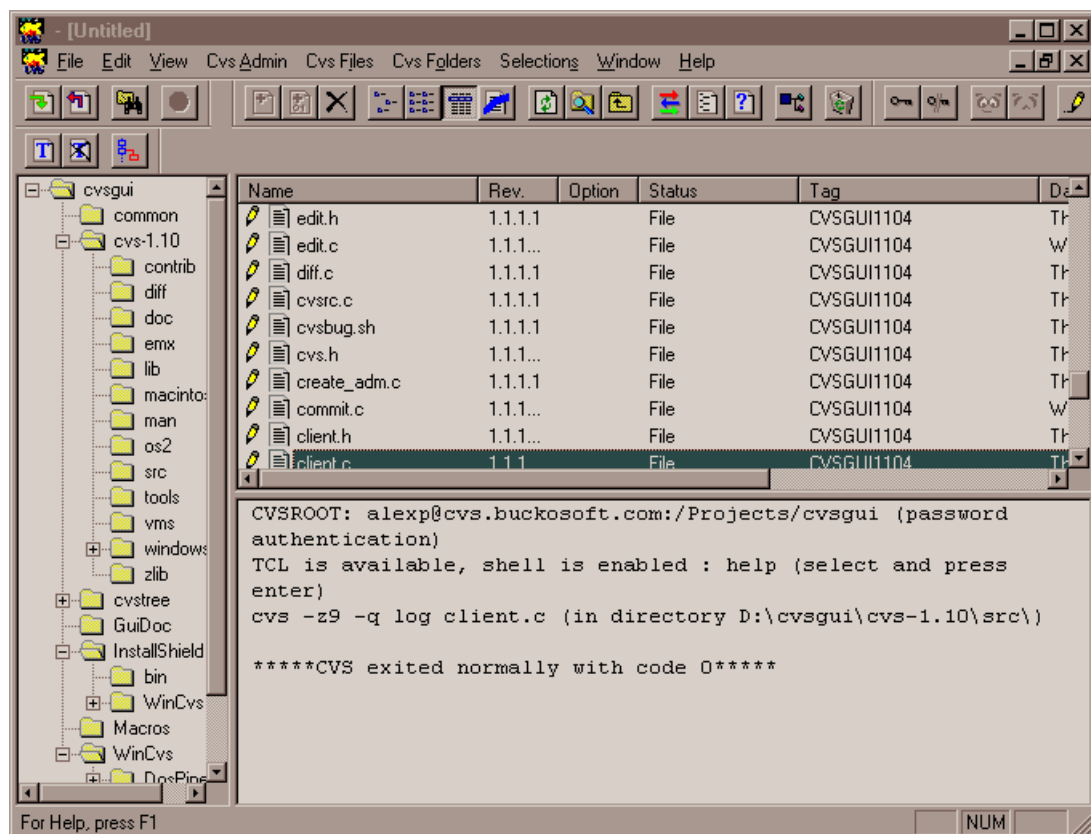


Figura 13.6: Um frontend para o cvs no Windows.

13.12 Um diagrama com os comandos do cvs

Veja na Figura ?? um diagrama com os comandos do cvs.

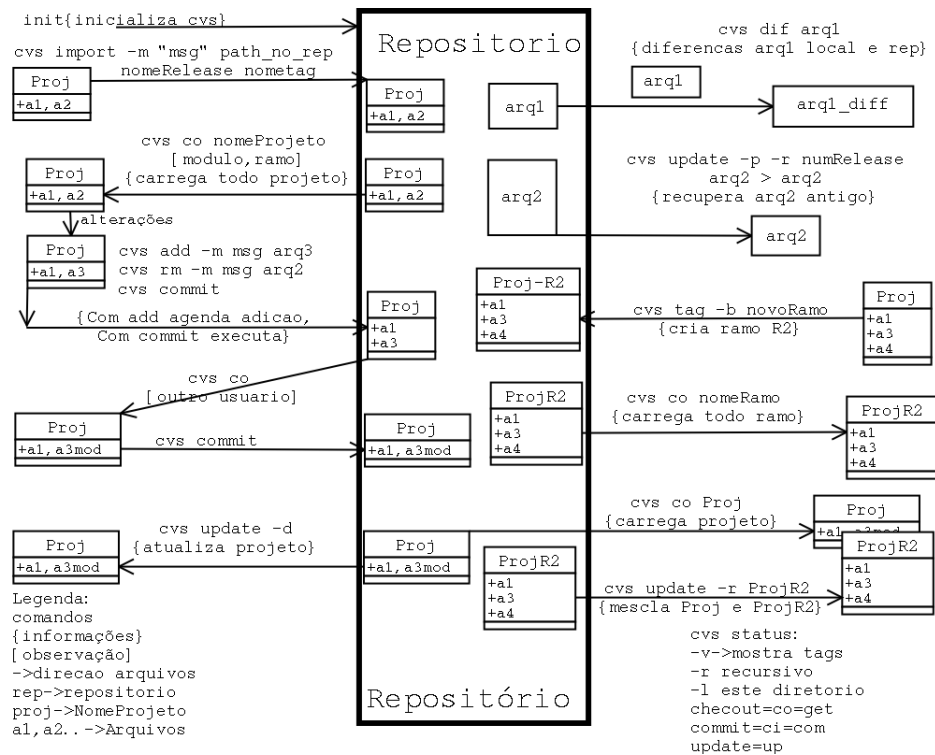


Figura 13.7: Diagrama com os comandos do cvs.

Referências Bibliográficas

- [1] Per Cederqvist. *Version Management with CVS*. GNU, 1993.
- [2] Mendel Cooper. *Building and Installing Software Packages for Linux - HOWTO*. LDP, 1999.
- [3] Hank Dietz. *Linux Parallel Processing HOWTO*. LDP, <http://yara.ecn.purdue.edu/ppli-nux/PPHOWTO/pphowto.html>, 1998.
- [4] Ricardo Gratti. *Bibliotecas Compartilhadas*, volume 5. Revista do Linux, 1999.
- [5] Ricardo Gratti. *Bibliotecas Compartilhadas*, volume 5. Revista do Linux, 1999.
- [6] Cameron Hughs and Tracey Hughes. *Object Oriented Multithreading using C++: architectures and components*, volume 1. John Wiley Sons, 2 edition, 1997.
- [7] Guilherme Wunsch Manika. *Super-Computador a Preço de Banana*, volume 2. Revista do Linux, 1999.
- [8] Ralf Nolden and Kdevelop-Team. *The User Manual to KDevelop*. kdevelop team, 1998.
- [9] Jacek Radajewski and Douglas Eadline. *Beowulf HOWTO*. LDP, <http://www.sci.usq.edu.au/staff/jacek/beowulf/BDP>, 1998.
- [10] Eric Steven Raymond. *Software Release Practice HOWTO*. LDP, 2000.
- [11] Alavoor Vasudevan. *C-C++ Beautifier HOWTO*. LDP, 2001.
- [12] Alavoor Vasudevan. *C++ Programming HOWTO*. LDP, 2001.
- [13] Kurt Wall. *Linux Programming Unleashed*, volume 1. SAMS, 2 edition, 2001.